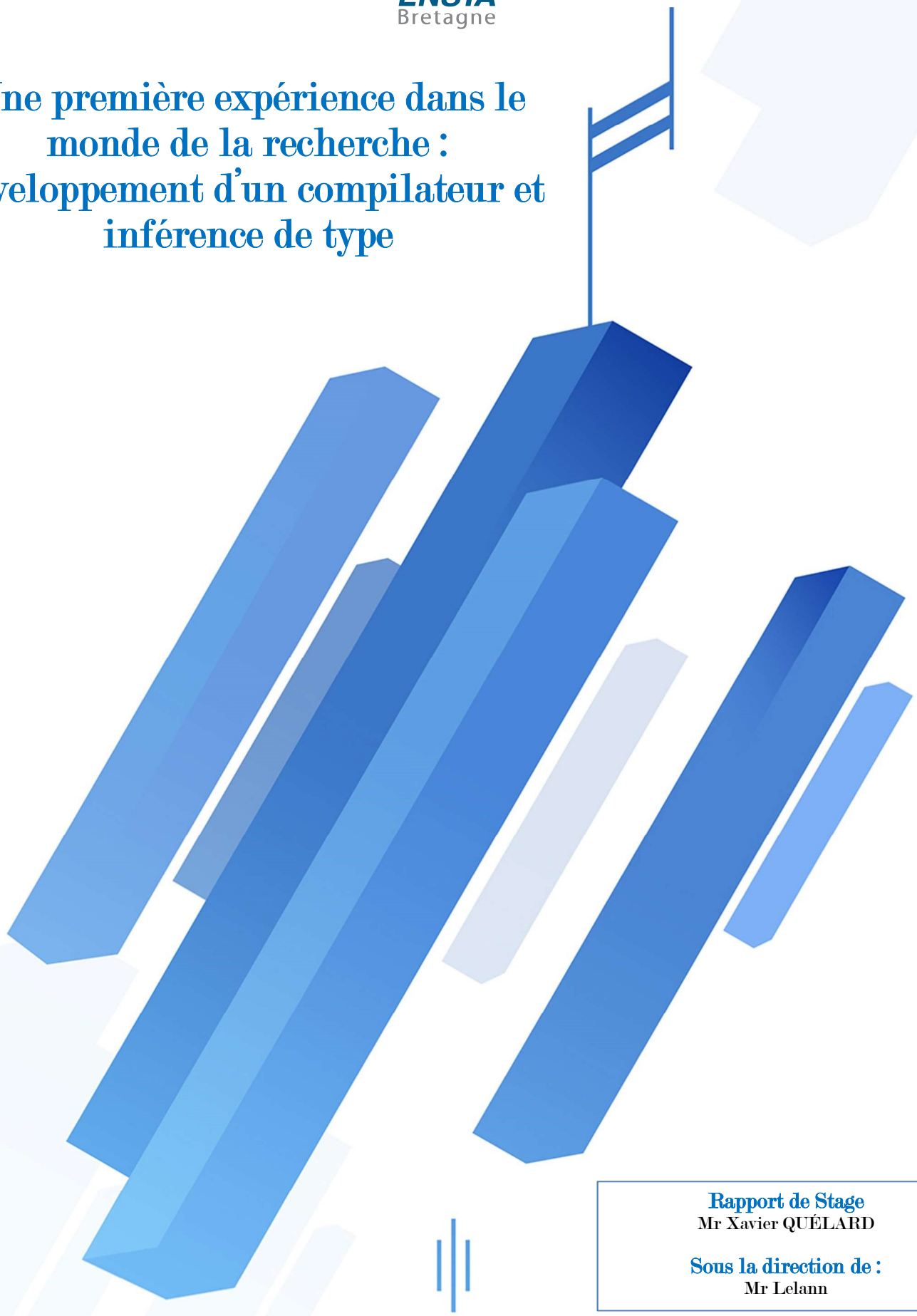
ENSTA Bretagne
Établissement d'enseignement supérieur public

2 rue François Verny
29200
Brest, France

Lab_STICC
Laboratoire des Sciences et Techniques de L'information, Communication et Connaissance

Technopôle Brest-Iroise
29200
Brest, France

Une première expérience dans le monde de la recherche : développement d'un compilateur et inférence de type



Rapport de Stage
Mr Xavier QUÉLARD

Sous la direction de :
Mr Lelann

Remerciements

```ruby
#this is a comment

a = 5
b = a * 5
puts b
puts b + 3

class Test < Test::SomeClass
@test = 3
    def bar
        if foo
            puts "foo"
        else
            puts "bar"
        end
    end
end
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eu arcu in lacus viverra fermentum et eget lectus. Suspendisse tortor orci, facilisis fringilla consequat et, scelerisque quis ipsum. Nulla accumsan lorem a malesuada ornare. Nunc pellentesque, mi commodo laoreet ullamcorper, nunc sem varius elit, hendrerit convallis risus lectus vitae mauris. Cras congue tincidunt felis, non molestie velit tempus non. Nam tempus convallis nunc, a accumsan ligula malesuada nec. Pellentesque urna velit, gravida nec tellus nec, placerat viverra sem. Quisque malesuada in justo at fringilla. Suspendisse lobortis iaculis dui non consectetur. Etiam lacinia laoreet arcu ac facilisis. Nunc ut vulputate ligula, non interdum libero. Nam in nunc at libero ullamcorper laoreet. Fusce quis nunc non tellus tincidunt vehicula.

Table des matières

Introduction

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eu arcu in lacus viverra fermentum et eget lectus. Suspendisse tortor orci, facilisis fringilla consequat et, scelerisque quis ipsum. Nulla accumsan lorem a malesuada ornare. Nunc pellentesque, mi commodo laoreet ullamcorper, nunc sem varius elit, hendrerit convallis risus lectus vitae mauris. Cras congue tincidunt felis, non molestie velit tempus non. Nam tempus convallis nunc, a accumsan ligula malesuada nec. Pellentesque urna velit, gravida nec tellus nec, placerat viverra sem. Quisque malesuada in justo at fringilla. Suspendisse lobortis iaculis dui non consectetur. Etiam lacinia laoreet arcu ac facilisis. Nunc ut vulputate ligula, non interdum libero. Nam in nunc at libero ullamcorper laoreet. Fusce quis nunc non tellus tincidunt vehicula.

Initiation au projet RubyESL

1.1 Contexte

1.1.1 High Level Syntesis

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eu arcu in lacus viverra fermentum et eget lectus. Suspendisse tortor orci, facilisis fringilla consequat et, scelerisque quis ipsum. Nulla accumsan lorem a malesuada ornare. Nunc pellentesque, mi commodo laoreet ullamcorper, nunc sem varius elit, hendrerit convallis risus lectus vitae mauris. Cras congue tincidunt felis, non molestie velit tempus non. Nam tempus convallis nunc, a accumsan ligula malesuada nec. Pellentesque urna velit, gravida nec tellus nec, placerat viverra sem. Quisque malesuada in justo at fringilla. Suspendisse lobortis iaculis dui non consectetur. Etiam lacinia laoreet arcu ac facilisis. Nunc ut vulputate ligula, non interdum libero. Nam in nunc at libero ullamcorper laoreet. Fusce quis nunc non tellus tincidunt vehicula.

1.1.2 SystemC et VIVADO

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eu arcu in lacus viverra fermentum et eget lectus. Suspendisse tortor orci, facilisis fringilla consequat et, scelerisque quis ipsum. Nulla accumsan lorem a malesuada ornare. Nunc pellentesque, mi commodo laoreet ullamcorper, nunc sem varius elit, hendrerit convallis risus lectus vitae mauris. Cras congue tincidunt felis, non molestie velit tempus non. Nam tempus convallis nunc, a accumsan ligula malesuada nec. Pellentesque urna velit, gravida nec tellus nec, placerat viverra sem. Quisque malesuada in justo at fringilla. Suspendisse lobortis iaculis dui non consectetur. Etiam lacinia laoreet arcu ac facilisis. Nunc ut vulputate ligula, non interdum libero. Nam in nunc at libero ullamcorper laoreet. Fusce quis nunc non tellus tincidunt vehicula.

1.2 Présentation du sujet proposé

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eu arcu in lacus viverra fermentum et eget lectus. Suspendisse tortor orci, facilisis fringilla consequat et, scelerisque quis ipsum. Nulla accumsan lorem a malesuada ornare. Nunc pellentesque, mi commodo laoreet ullamcorper, nunc sem varius elit, hen-

drerit convallis risus lectus vitae mauris. Cras congue tincidunt felis, non molestie velit tempus non. Nam tempus convallis nunc, a accumsan ligula malesuada nec. Pellentesque urna velit, gravida nec tellus nec, placerat viverra sem. Quisque malesuada in justo at fringilla. Suspendisse lobortis iaculis dui non consectetur. Etiam lacinia laoreet arcu ac facilisis. Nunc ut vulputate ligula, non interdum libero. Nam in nunc at libero ullamcorper laoreet. Fusce quis nunc non tellus tincidunt vehicula.

1.3 Etat de l'art

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eu arcu in lacus viverra fermentum et eget lectus. Suspendisse tortor orci, facilisis fringilla consequat et, scelerisque quis ipsum. Nulla accumsan lorem a malesuada ornare. Nunc pellentesque, mi commodo laoreet ullamcorper, nunc sem varius elit, hendrerit convallis risus lectus vitae mauris. Cras congue tincidunt felis, non molestie velit tempus non. Nam tempus convallis nunc, a accumsan ligula malesuada nec. Pellentesque urna velit, gravida nec tellus nec, placerat viverra sem. Quisque malesuada in justo at fringilla. Suspendisse lobortis iaculis dui non consectetur. Etiam lacinia laoreet arcu ac facilisis. Nunc ut vulputate ligula, non interdum libero. Nam in nunc at libero ullamcorper laoreet. Fusce quis nunc non tellus tincidunt vehicula.

Conduite du projet

2.1 Choix technologiques

2.1.1 Langage Ruby

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eu arcu in lacus viverra fermentum et eget lectus. Suspendisse tortor orci, facilisis fringilla consequat et, scelerisque quis ipsum. Nulla accumsan lorem a malesuada ornare. Nunc pellentesque, mi commodo laoreet ullamcorper, nunc sem varius elit, hendrerit convallis risus lectus vitae mauris. Cras congue tincidunt felis, non molestie velit tempus non. Nam tempus convallis nunc, a accumsan ligula malesuada nec. Pellentesque urna velit, gravida nec tellus nec, placerat viverra sem. Quisque malesuada in justo at fringilla. Suspendisse lobortis iaculis dui non consectetur. Etiam lacinia laoreet arcu ac facilisis. Nunc ut vulputate ligula, non interdum libero. Nam in nunc at libero ullamcorper laoreet. Fusce quis nunc non tellus tincidunt vehicula.

2.1.2 Librairies et logiciels

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eu arcu in lacus viverra fermentum et eget lectus. Suspendisse tortor orci, facilisis fringilla consequat et, scelerisque quis ipsum. Nulla accumsan lorem a malesuada ornare. Nunc pellentesque, mi commodo laoreet ullamcorper, nunc sem varius elit, hendrerit convallis risus lectus vitae mauris. Cras congue tincidunt felis, non molestie velit tempus non. Nam tempus convallis nunc, a accumsan ligula malesuada nec. Pellentesque urna velit, gravida nec tellus nec, placerat viverra sem. Quisque malesuada in justo at fringilla. Suspendisse lobortis iaculis dui non consectetur. Etiam lacinia laoreet arcu ac facilisis. Nunc ut vulputate ligula, non interdum libero. Nam in nunc at libero ullamcorper laoreet. Fusce quis nunc non tellus tincidunt vehicula.

2.2 Organisation du travail

2.2.1 Horaires, réunions et temps de travail

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eu arcu in lacus viverra fermentum et eget lectus. Suspendisse tortor orci, facilisis fringilla consequat et, scelerisque quis ipsum. Nulla accumsan lorem a malesuada ornare.

Nunc pellentesque, mi commodo laoreet ullamcorper, nunc sem varius elit, hendrerit convallis risus lectus vitae mauris. Cras congue tincidunt felis, non molestie velit tempus non. Nam tempus convallis nunc, a accumsan ligula malesuada nec. Pellentesque urna velit, gravida nec tellus nec, placerat viverra sem. Quisque malesuada in justo at fringilla. Suspendisse lobortis iaculis dui non consectetur. Etiam lacinia laoreet arcu ac facilisis. Nunc ut vulputate ligula, non interdum libero. Nam in nunc at libero ullamcorper laoreet. Fusce quis nunc non tellus tincidunt vehicula.

2.2.2 Diagramme de GANTT

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eu arcu in lacus viverra fermentum et eget lectus. Suspendisse tortor orci, facilisis fringilla consequat et, scelerisque quis ipsum. Nulla accumsan lorem a malesuada ornare. Nunc pellentesque, mi commodo laoreet ullamcorper, nunc sem varius elit, hendrerit convallis risus lectus vitae mauris. Cras congue tincidunt felis, non molestie velit tempus non. Nam tempus convallis nunc, a accumsan ligula malesuada nec. Pellentesque urna velit, gravida nec tellus nec, placerat viverra sem. Quisque malesuada in justo at fringilla. Suspendisse lobortis iaculis dui non consectetur. Etiam lacinia laoreet arcu ac facilisis. Nunc ut vulputate ligula, non interdum libero. Nam in nunc at libero ullamcorper laoreet. Fusce quis nunc non tellus tincidunt vehicula.

2.3 Structure de la solution proposée

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eu arcu in lacus viverra fermentum et eget lectus. Suspendisse tortor orci, facilisis fringilla consequat et, scelerisque quis ipsum. Nulla accumsan lorem a malesuada ornare. Nunc pellentesque, mi commodo laoreet ullamcorper, nunc sem varius elit, hendrerit convallis risus lectus vitae mauris. Cras congue tincidunt felis, non molestie velit tempus non. Nam tempus convallis nunc, a accumsan ligula malesuada nec. Pellentesque urna velit, gravida nec tellus nec, placerat viverra sem. Quisque malesuada in justo at fringilla. Suspendisse lobortis iaculis dui non consectetur. Etiam lacinia laoreet arcu ac facilisis. Nunc ut vulputate ligula, non interdum libero. Nam in nunc at libero ullamcorper laoreet. Fusce quis nunc non tellus tincidunt vehicula.

Réalisation technique

3.0.1 Application du pattern Factory au typage

L'une des différences clés entre Ruby et SystemC, comme vu précédemment, est le typage des variables. Là où le C++ (et donc le SystemC, par extension) est un langage au typage statique[1], il est difficile de ne serais-ce que parler de notion de type en Ruby. En réalité, toutes les variables en ruby ne sont que des objets : la notion de type n'a donc en soi aucun sens. Pour déterminer si une variable est "compatible" ou non pour une opération donnée, Ruby va donc utiliser la philosophie du *Duck Typing*[2] (littéralement : le "typage-canard") : "si un objet marche, mange et crie comme un canard, alors c'est un canard". L'idée est donc de s'intéresser aux méthodes et attributs composant l'objet, plutôt qu'à sa nature profonde (Un exemple est donné en annexe, p26). Cela rends notamment le polymorphisme en Ruby extrêmement naturel.

Lors du développement de la solution RubyESL, l'idée de créer des classes `Type` est rapidement apparue comme nécessaire. Ont ainsi été écrites, dans le fichier mts_types.rb, cinq classes :

- Une superclasse abstraite `Type`
- Une classe `SingleType`, héritant de `Type`, et représentant un type "simple" (int, float..)
- Une classe `UnionType`, héritant de `Type`, et représentant une variable pouvant avoir plusieurs types distincs (exemple : Float | Int)
- Une classe `ArrayType`, héritant de `Type`, et représentant le type d'un array ([Int], [Float | Int], ...)
- Une classe `TypeFactory`, qui permet d'associer à une variable ruby un `Type`

1. Une variable est initialisée avec un type donné, et ne peut plus en changer pendant toute l'éxécution du programme

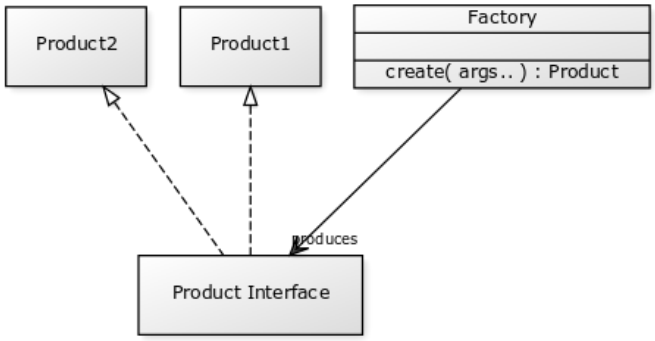


FIGURE 3.1 – Diagramme UML du design pattern *Factory*.

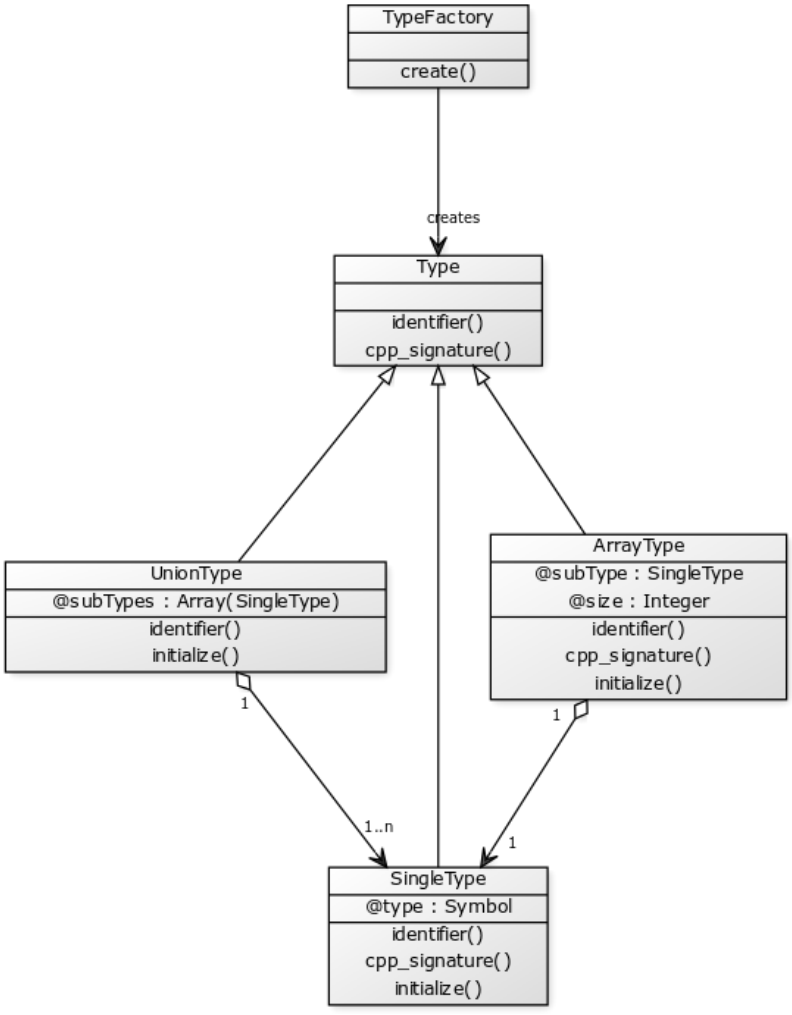


FIGURE 3.2 – Diagramme UML des classes de typage contenues dans le fichier mts_types.rb .

Pour la classe `TypeFactory`, le design pattern *Factory* [3.2, p11] est utilisé. Une classe `Factory` a pour unique rôle de créer des objets dont la classe hérite

de `Product`. Ici, la classe `TypeFactory`[2] génèrera donc un objet `Type` différent (single, union ou array) en fonction de la variable d'entrée.

3.0.2 Etablissement d'un DSL

Un DSL[4] (pour Domain Specific programming Language) est un langage de programmation dont les spécifications sont concues pour répondre aux contraintes d'un domaine d'applications précis. Ainsi, SystemC est un DSL interne à C++, qui permet de définir de manière plus simple un système hardware. Il semblait ainsi naturel de créer un tel dsl en Ruby pour décrire nos propres systèmes. Le fichier mts_dsl.rb décrit ce dsl, nommé *RubyESL*. Voici un exemple de la création d'un acteur (l'équivalent d'un module en SystemC) :

```
class Sinker < Actor
  input  :outp
  thread :sink

  def sink
    for k in 0...64
      datain = read(:outp)
      wait()
    end
    stop()
  end

end
```

Dans l'exemple ci dessus, hériter de la classe Actor permet à la classe Sinker de définir un acteur, et d'avoir accès aux méthodes :

- `read(:port)` correspondant à la fonction `port.read()` de SystemC[3]
- `write(:port, val)` correspondant à la fonction `port.write(val)` de SystemC
- `wait()` correspondant à la fonction `wait()` de SystemC
- `stop()` correspondant à la fonction `stop\_simulation()` de SystemC

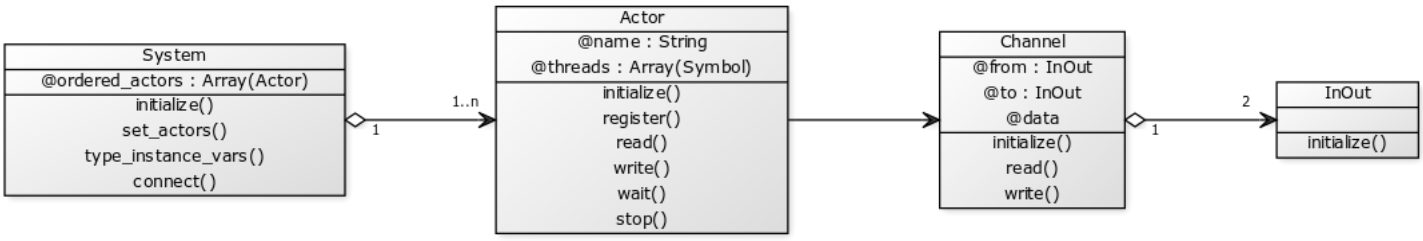


FIGURE 3.3 – Diagramme UML du DSL réalisé.

2. Je parle bien ici de classe et non pas d'objet. En effet, la methode `create()` de la classe `TypeFactory` est une méthode de classe, et non pas une méthode d'instance.
3. Par soucis de lisibilité, je confonds ici méthode et exemple appel de celle-ci.

3.1 Parsing du code d'entrée

3.1.1 Whitequark

Comme vu dans la partie 2.3, la première étape nécessaire à la réalisation d'un compilateur est l'analyse syntaxique (ou parsing, en anglais) du code source d'entrée. Il s'agit d'une étape assez fastidieuse et répétitive. Pour ces raisons, il me semblait vain de vouloir ré-inventer la roue : j'ai donc choisi d'utiliser une librarie externe pour s'occuper du parsing, afin de pouvoir consacrer plus de temps aux étapes suivantes, plus pédagogiques.

La librairie proposée est Parser de Whitequark[3], pour les raisons suivantes :
— Il est aussi complet que le parser officiel, du nom de Ripper[1] ;
— Il est production ready ;
— Il offre une série entière d'outils pour exploiter le code parsé (qui seront utilisés dans l'étape suivante [3.2.1, p15]).

En plus de ces qualités, le parser est entièrement écrit en Ruby, ce qui est un bonus appréciable.

La parser de Whitequark renvoie le code parsé sous forme de *s-expressions*[5] : il s'agit d'objets possédant un identifiant (un symbole précisant quel type de node est représenté), et optionnellement un ou plusieurs paramètres et enfants(qui sont également des s-expressions). Ainsi, le code ruby `i = 2+3` sera converti en la s-expression suivante :

```
s(:lvasgn, :i,
  s(:send,
    s(:int, 2), :+,
    s(:int, 3)
  )
)
```

Ici, la s-expression d'identifiant `:lvasgn` (assignation de variable) a pour paramètre `:i`, et pour enfant la s-expression `:send` (utilisation d'une méthode). La s-expression `:send` a quant à elle deux enfants `:int`, qui sont les valeurs 2 et 3. Le nom de la méthode appelée (ici, il s'agit de l'addition `:+`) est également précisé.

3.1.2 Objectification

Le compilateur *RubyESL* prends ensuite le code parsé en entrée et effectue une "objectification" de ce dernier. Il s'agit d'extraire le contenu des s-expression et de les stocker dans différents objets container. En effet, dans l'étape finale qu'est la génération du code SystemC [3.3, p18] , le pattern *Visitor* est utilisé. Pour cela, les objets à parcourir (ici, notre code, donc nos s-expressions) doivent posséder une méthode leur permettant d'être visitées. Réorganiser les s-expressions du code de départ dans des container permet ainsi de faire fonctionner le pattern *Visitor*.

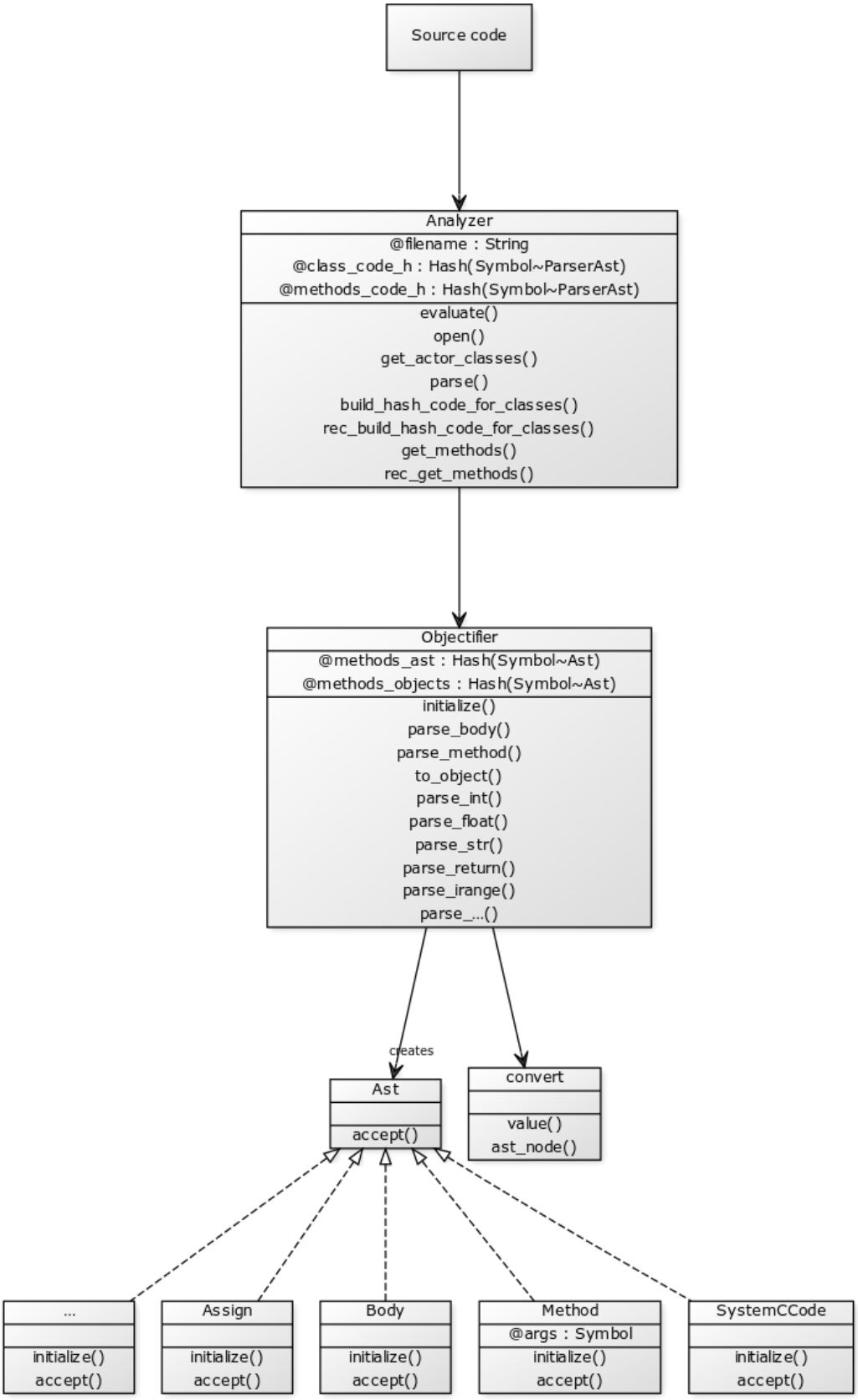


FIGURE 3.4 – Diagramme UML de la partie "objectification" du compilateur RubyESL, disponible dans les fichiers mts_objects.rb et mts_objectifier.rb

Mais l'étape de l'objectification n'est pas uniquement un ré-écriture de l'information présente. C'est également l'occasion de contextualiser le code : ainsi, de nouveaux attributs sont ajoutés aux containers. Voici un exemple de cet ajout d'information :

```
Class Analyzer
    def evaluate filename
      rcode=IO.read(filename)
      eval(rcode)
    end

    ...
end
```

Lors du parsing de ce code, une s-expression `:def` va être créé au moment du parcours de la définition de la méthode `evaluate`. L'objectifier va donc créer un objet de la classe `Method`, contenant les même informations que celles de la s-expression (son nom, ses arguments, et son contenu). Cependant, au moment de la génération du code SystemC, chaque méthode doit déclarer ses variables internes (avec leur signature), chose qu'une méthode en ruby n'a pas besoin de faire. Un attribut `@toDeclare` est donc ajouté à l'objet représentant la méthod evaluate, qui contient l'ensemble des variables à déclarer (ici,).

Avant d'être sauvegardés, les objets traduisant le code source de départ subissent une dernière transformation : ils sont passé en argument à une méthode de conversion. Cette dernière examine l'objet en question, et le transforme si besoin. L'objectif, ici, est de gérer les conversions à faire entre Ruby et SystemC. Par exemple, la méthode pour afficher du texte en Ruby est `puts`, tandis qu'on utilise `cout` en SystemC. La méthode de conversion va ainsi changer l'attribut `@name` de `:puts` à `:cout`.

Une autre conversion nécessaire est celle des opérateurs. En effet, les opérateurs de comparaison (`<`, `>`, `<=`, `>=`, `==`) sont considérés comme de simples méthodes en Ruby. En revanche, pour le C++, ce sont des caractères spéciaux réservés pour cet unique usage. L'objet de la classe `Method` va alors être converti en un objet de la classe `Operator`, qui sera interprêté et traduit correctement par le Visitor.

3.1.3 Bilan

Un exemple de parsing puis objectification d'un code source est donné en annexe (voir annexe, p24).

3.2 Typage dynamique des variables

3.2.1 Ajout de probes

La suite logique du processus est de simuler le code Ruby entré par l'utilisateur, afin d'extrapoler le type des variables le composant, ainsi que le type de retour

des méthodes. Pour faciliter grandement le processus, une étape intermédiaire est effectué : l'ajout de "probes" dans le code source d'entrée. Ainsi, après chaque ligne de code utilisant une variable, une ligne de code supplémentaire est ajoutée, pour indiquer au simulateur que la variable en question a été utilisée, et lui transmettre sa nouvelle valeur. De cette manière, le simulateur peur facilement traquer les variables des code "probés". Ci-dessous, un exemple de deux blocs de code, avant et après l'ajout des probes.

```
  def initialize name, ucoef
    @coef = [0,0,0,0,0]

    for i in 0...5
      @coef[i] = ucoef[i]
    end

    super(name, ucoef)
  end

# ============================================================

  def initialize name, ucoef
    @coef = [0,0,0,0,0]
    register(:@coef, tmp, self.class.get_klass(), __method__)

    for i in 0...5
      @coef[i] = ucoef[i]
      register(:@coef, tmp, self.class.get_klass(), __method__)
    end

    super(name, ucoef)
    register(:ucoef, tmp, self.class.get_klass(), __method__)
  end
```

Un appel à la méthode `register` du simulateur permet de mettre à jour le type d'une variable. On remarque sont passés en paramètres :

- le nom de la variable
- le contenu de la variable
- le nom de la classe de l'objet dans laquelle elle vit
- le nom de la méthode dans laquelle elle vit

Ces quatre informations suffisent au simulateur pour savoir précisément de quelle variable il s'agit, et quel est le "type" de son contenu (la section suivante détaille cette notion de type).

3.2.2 Simulation et application du pattern Singleton

L'ultime étape avant la ré-écriture du code d'entrée en SystemC est, comme vu précédemment, la simulation de ce dernier, afin de pouvoir inférer les types des

variables et ainsi générer du C++.Il fut donc nécessaire de réaliser un simulateur de code RubyESL. La classe Fiber de Ruby permet alors de réaliser sans difficulté un tel simulateur : un objet `Fiber` prends en entrée un block de code ruby, et peut l'éxécuter partiellement ou totalement par appel de méthodes. La classe simulateur possède ainsi les attributs et méthodes suivantes :

- `@fibers` qui est un tableau contenant l'ensemble de nos objets de la classe `Fiber` ;
- `@time` qui contient un entier naturel représentant le nombre de tours de simulation effectués ;
- `@muststop` booléen permettant de gérer l'arrêt du simulateur en cours de route ;
- `add_fiber()` qui permet d'ajouter un bloc de code à simuler ;
- `keep_going()` qui permet de checker la valeur de `@muststop` ;
- `stop()` qui permet de forcer l'arrêt de la simulation depuis l'intérieur d'un code simulé ;
- `step()` qui permet d'avancer d'un pas supplémentaire dans la simulation de toutes les Fibers ;
- `run()` qui permet de lancer la simulation.

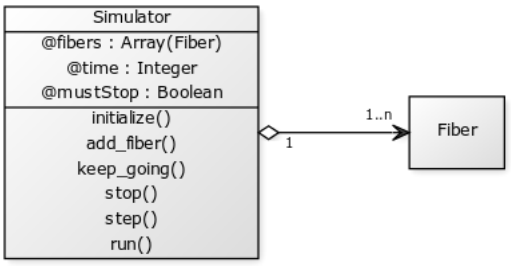


FIGURE 3.5 – Diagramme UML du simulateur réalisé.

Comme vu dans la section concernant le DSL RubyESL, il est mis à disposition des classes simulées une méthode `stop()` permettant d'arrêter la simulation. Pour faire le lien entre le DSL et le simulateur, il était nécessaire que chaque acteur garde une référence du simulateur qui est en train de les éxécuter. Pour résoudre ce type de problème d'accessibilité, une classe `Globaldata` contenant les informations dont l'accès doit être possible dans tout l'environnement du programme a été écrite. Celle-ci suit le design pattern *Singleton*.

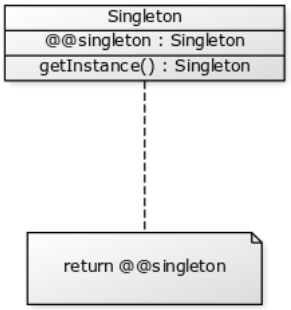


FIGURE 3.6 – Diagramme UML du design pattern `Singleton`.

Le but est d'obtenir un unique objet (dans le cas présent, un objet `DATA` contenant toutes les informations globales du système) héritant de la classe `Singleton`. La particularité de cette dernière est qu'une seule et unique instance est permise, et n'est pas transmise via l'utilisation de `Singleton.new`. On appelle à la place la méthode de classe `Singleton.getInstance()`. Par l'utilisation d'une variable de classe compteur d'instance, le pattern garanti l'unicité de l'objet obtenu.

3.3 Generation de code SystemC

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eu arcu in lacus viverra fermentum et eget lectus. Suspendisse tortor orci, facilisis fringilla consequat et, scelerisque quis ipsum. Nulla accumsan lorem a malesuada ornare. Nunc pellentesque, mi commodo laoreet ullamcorper, nunc sem varius elit, hendrerit convallis risus lectus vitae mauris. Cras congue tincidunt felis, non molestie velit tempus non. Nam tempus convallis nunc, a accumsan ligula malesuada nec. Pellentesque urna velit, gravida nec tellus nec, placerat viverra sem. Quisque malesuada in justo at fringilla. Suspendisse lobortis iaculis dui non consectetur. Etiam lacinia laoreet arcu ac facilisis. Nunc ut vulputate ligula, non interdum libero. Nam in nunc at libero ullamcorper laoreet. Fusce quis nunc non tellus tincidunt vehicula.

Experimentations, mesures et résultats

4.1 Exemple d'application : filtre Fir

4.1.1 Présentation du filtre

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eu arcu in lacus viverra fermentum et eget lectus. Suspendisse tortor orci, facilisis fringilla consequat et, scelerisque quis ipsum. Nulla accumsan lorem a malesuada ornare. Nunc pellentesque, mi commodo laoreet ullamcorper, nunc sem varius elit, hendrerit convallis risus lectus vitae mauris. Cras congue tincidunt felis, non molestie velit tempus non. Nam tempus convallis nunc, a accumsan ligula malesuada nec. Pellentesque urna velit, gravida nec tellus nec, placerat viverra sem. Quisque malesuada in justo at fringilla. Suspendisse lobortis iaculis dui non consectetur. Etiam lacinia laoreet arcu ac facilisis. Nunc ut vulputate ligula, non interdum libero. Nam in nunc at libero ullamcorper laoreet. Fusce quis nunc non tellus tincidunt vehicula.

4.1.2 Transcription du filtre en SystemC

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eu arcu in lacus viverra fermentum et eget lectus. Suspendisse tortor orci, facilisis fringilla consequat et, scelerisque quis ipsum. Nulla accumsan lorem a malesuada ornare. Nunc pellentesque, mi commodo laoreet ullamcorper, nunc sem varius elit, hendrerit convallis risus lectus vitae mauris. Cras congue tincidunt felis, non molestie velit tempus non. Nam tempus convallis nunc, a accumsan ligula malesuada nec. Pellentesque urna velit, gravida nec tellus nec, placerat viverra sem. Quisque malesuada in justo at fringilla. Suspendisse lobortis iaculis dui non consectetur. Etiam lacinia laoreet arcu ac facilisis. Nunc ut vulputate ligula, non interdum libero. Nam in nunc at libero ullamcorper laoreet. Fusce quis nunc non tellus tincidunt vehicula.

4.2 Exemple d'application : autre application

4.2.1 Présentation de l'application

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eu arcu in

lacus viverra fermentum et eget lectus. Suspendisse tortor orci, facilisis fringilla consequat et, scelerisque quis ipsum. Nulla accumsan lorem a malesuada ornare. Nunc pellentesque, mi commodo laoreet ullamcorper, nunc sem varius elit, hendrerit convallis risus lectus vitae mauris. Cras congue tincidunt felis, non molestie velit tempus non. Nam tempus convallis nunc, a accumsan ligula malesuada nec. Pellentesque urna velit, gravida nec tellus nec, placerat viverra sem. Quisque malesuada in justo at fringilla. Suspendisse lobortis iaculis dui non consectetur. Etiam lacinia laoreet arcu ac facilisis. Nunc ut vulputate ligula, non interdum libero. Nam in nunc at libero ullamcorper laoreet. Fusce quis nunc non tellus tincidunt vehicula.

4.2.2 Transcription de l'application en SystemC

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eu arcu in lacus viverra fermentum et eget lectus. Suspendisse tortor orci, facilisis fringilla consequat et, scelerisque quis ipsum. Nulla accumsan lorem a malesuada ornare. Nunc pellentesque, mi commodo laoreet ullamcorper, nunc sem varius elit, hendrerit convallis risus lectus vitae mauris. Cras congue tincidunt felis, non molestie velit tempus non. Nam tempus convallis nunc, a accumsan ligula malesuada nec. Pellentesque urna velit, gravida nec tellus nec, placerat viverra sem. Quisque malesuada in justo at fringilla. Suspendisse lobortis iaculis dui non consectetur. Etiam lacinia laoreet arcu ac facilisis. Nunc ut vulputate ligula, non interdum libero. Nam in nunc at libero ullamcorper laoreet. Fusce quis nunc non tellus tincidunt vehicula.

4.3 Benchmarks

4.3.1 Filtre FIR

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eu arcu in lacus viverra fermentum et eget lectus. Suspendisse tortor orci, facilisis fringilla consequat et, scelerisque quis ipsum. Nulla accumsan lorem a malesuada ornare. Nunc pellentesque, mi commodo laoreet ullamcorper, nunc sem varius elit, hendrerit convallis risus lectus vitae mauris. Cras congue tincidunt felis, non molestie velit tempus non. Nam tempus convallis nunc, a accumsan ligula malesuada nec. Pellentesque urna velit, gravida nec tellus nec, placerat viverra sem. Quisque malesuada in justo at fringilla. Suspendisse lobortis iaculis dui non consectetur. Etiam lacinia laoreet arcu ac facilisis. Nunc ut vulputate ligula, non interdum libero. Nam in nunc at libero ullamcorper laoreet. Fusce quis nunc non tellus tincidunt vehicula.

4.3.2 Autre application

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eu arcu in lacus viverra fermentum et eget lectus. Suspendisse tortor orci, facilisis fringilla consequat et, scelerisque quis ipsum. Nulla accumsan lorem a malesuada ornare.

Nunc pellentesque, mi commodo laoreet ullamcorper, nunc sem varius elit, hendrerit convallis risus lectus vitae mauris. Cras congue tincidunt felis, non molestie velit tempus non. Nam tempus convallis nunc, a accumsan ligula malesuada nec. Pellentesque urna velit, gravida nec tellus nec, placerat viverra sem. Quisque malesuada in justo at fringilla. Suspendisse lobortis iaculis dui non consectetur. Etiam lacinia laoreet arcu ac facilisis. Nunc ut vulputate ligula, non interdum libero. Nam in nunc at libero ullamcorper laoreet. Fusce quis nunc non tellus tincidunt vehicula.

Bilan

5.1 Bilan technique

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eu arcu in lacus viverra fermentum et eget lectus. Suspendisse tortor orci, facilisis fringilla consequat et, scelerisque quis ipsum. Nulla accumsan lorem a malesuada ornare. Nunc pellentesque, mi commodo laoreet ullamcorper, nunc sem varius elit, hendrerit convallis risus lectus vitae mauris. Cras congue tincidunt felis, non molestie velit tempus non. Nam tempus convallis nunc, a accumsan ligula malesuada nec. Pellentesque urna velit, gravida nec tellus nec, placerat viverra sem. Quisque malesuada in justo at fringilla. Suspendisse lobortis iaculis dui non consectetur. Etiam lacinia laoreet arcu ac facilisis. Nunc ut vulputate ligula, non interdum libero. Nam in nunc at libero ullamcorper laoreet. Fusce quis nunc non tellus tincidunt vehicula.

5.2 Bilan personnel et professionnel

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eu arcu in lacus viverra fermentum et eget lectus. Suspendisse tortor orci, facilisis fringilla consequat et, scelerisque quis ipsum. Nulla accumsan lorem a malesuada ornare. Nunc pellentesque, mi commodo laoreet ullamcorper, nunc sem varius elit, hendrerit convallis risus lectus vitae mauris. Cras congue tincidunt felis, non molestie velit tempus non. Nam tempus convallis nunc, a accumsan ligula malesuada nec. Pellentesque urna velit, gravida nec tellus nec, placerat viverra sem. Quisque malesuada in justo at fringilla. Suspendisse lobortis iaculis dui non consectetur. Etiam lacinia laoreet arcu ac facilisis. Nunc ut vulputate ligula, non interdum libero. Nam in nunc at libero ullamcorper laoreet. Fusce quis nunc non tellus tincidunt vehicula.

Conclusion

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eu arcu in lacus viverra fermentum et eget lectus. Suspendisse tortor orci, facilisis fringilla consequat et, scelerisque quis ipsum. Nulla accumsan lorem a malesuada ornare. Nunc pellentesque, mi commodo laoreet ullamcorper, nunc sem varius elit, hendrerit convallis risus lectus vitae mauris. Cras congue tincidunt felis, non molestie velit tempus non. Nam tempus convallis nunc, a accumsan ligula malesuada nec. Pellentesque urna velit, gravida nec tellus nec, placerat viverra sem. Quisque malesuada in justo at fringilla. Suspendisse lobortis iaculis dui non consectetur. Etiam lacinia laoreet arcu ac facilisis. Nunc ut vulputate ligula, non interdum libero. Nam in nunc at libero ullamcorper laoreet. Fusce quis nunc non tellus tincidunt vehicula.

Annexe

5.3 Parsing et Objectification d'un code source

Code de départ :

```
def source
    tmp = 0
    for i in 0...64
      if (i > 23 && i < 29)
        tmp = 256
      else
        tmp = 0
      end

      write(tmp, :inp)
      wait()
    end
end
```

Après parsing :

```
s(:begin,
  s(:send, nil, :puts,
  s(:lvasgn, :tmp,
    s(:int, 0)),
  s(:for,
    s(:lvasgn, :i),
    s(:erange,
      s(:int, 0),
      s(:int, 64)),
    s(:begin,
      s(:if,
        s(:begin,
          s(:and,
            s(:send,
              s(:lvar, :i), :>,
              s(:int, 23)),
            s(:send,
```

```
            s(:lvar, :i), :<,
            s(:int, 29)))),
      s(:lvasgn, :tmp,
        s(:int, 256)),
      s(:lvasgn, :tmp,
        s(:int, 0))),
    s(:send, nil, :write,
      s(:lvar, :tmp),
      s(:sym, :inp)),
    s(:send, nil, :wait))))
```

Puis après conversion en containers :

```
#<NMTS::Method:0x000000000362d948
      @args=[],
      @body=
       #<NMTS::Body:0x000000000362d998
        @stmts=
         [#<NMTS::Assign:0x00000000033d3670
           @lhs=:tmp,
           @rhs=#<NMTS::IntLit:0x00000000033d3800 @value=0>>,
          #<NMTS::For:0x000000000362dbf0
           @body=
            #<NMTS::Body:0x00000000035bbb68
             @stmts=
              [#<NMTS::If:0x000000000305b240
                @body=
                 #<NMTS::Body:0x000000000305b470
                  @stmts=
                   [#<NMTS::Assign:0x000000000305b560
                     @lhs=:tmp,
                     @rhs=#<NMTS::IntLit:0x00000000033b5e68 @value=256>>],
                  @toDeclare={},
                  @wrapperBody=true>,
                @cond=
                 #<NMTS::Body:0x00000000033b60e8
                  @stmts=
                   [#<NMTS::And:0x00000000033b62c8
                     @lhs=
                      #<NMTS::SystemCCode:0x00000000033bffa8 @code="i > 23">,
                     @rhs=
                      #<NMTS::SystemCCode:0x00000000033b62f0
                       @code="i < 29">>],
                  @toDeclare={},
                  @wrapperBody=false>,
                @else_=
                 #<NMTS::Body:0x000000000305b2b8
```

```
                    @stmts=
                     [#<NMTS::Assign:0x000000000305b358
                       @lhs=:tmp,
                       @rhs=#<NMTS::IntLit:0x00000000033b58a0 @value=0>>],
                    @toDeclare={},
                    @wrapperBody=true>>,
                  #<NMTS::SystemCCode:0x0000000003041cc8
                   @code="inp.write(tmp)">,
                  #<NMTS::MCall:0x0000000003041c00
                   @args=[],
                   @caller=nil,
                   @method=:wait>],
                 @toDeclare={},
                 @wrapperBody=true>,
               @cond=#<NMTS::Assign:0x00000000033d3418 @lhs=:i, @rhs=nil>,
               @range=
                #<NMTS::ERange:0x00000000033d2e28
                 @idx=:i,
                 @lhs=#<NMTS::IntLit:0x00000000033d31c0 @value=0>,
                 @rhs=#<NMTS::IntLit:0x00000000033d3030 @value=64>>>],
             @toDeclare={},
             @wrapperBody=true>,
           @name=:source,
           @returnType=#<NMTS::SingleType:0x000000000362d8a8 @type=NilClass>,
           @toDeclare={}>}]
```

5.4 Duck typing en ruby

```ruby
# plan de fabrication d'un "vrai" canard
Class Canard
  def coin_coin
    puts "coin coin"
  end
end

# plan de fabrication d'un "faux" canard
Class RobotCanard
  def coin_coin
    puts "koin... kkoin"
  end
end

class DetecteurCanard
  def self.faire_coin_coin objet
    # peu importe la classe de l'objet passé en paramètre
    # s'il possède une méthode coin_coin, le code s'éxécute
```

```
    objet.coin_coin
  end
end

vrai_canard = Canard.new
faux_canard = RobotCanard.new

# les deux fonctionnent
DetecteurCanard.faire_coin_coin vrai_canard # 'coin coin'
DetecteurCanard.faire_coin_coin faux_canard # 'koin... kkoin'
```

Bibliographie

[1] Doc Ruby. http ://ruby-doc.org/stdlib-2.0.0/libdoc/ripper/rdoc/ripper.html - ripper is a ruby script parser.

[2] RubyLearning. http ://rubylearning.com/satishtalim/duck_typing.html - duck typing.

[3] Whitequark. https ://github.com/whitequark/parser - production-ready ruby parser written in pure ruby.

[4] Wikipédia. https ://fr.wikipedia.org/wiki/langage_d

[5] Wikipédia. https ://fr.wikipedia.org/wiki/s-expression - s-expression.