Migration de Fortran/ESOPE vers Fortran-2003

Younoussa Sow

1 Février - 30 Juin 2022

Tuteur école :
Frédéric Hoogstoel

Tuteur organisme :
Nicolas Anquetil

Résumé

Ce rapport retrace mes 5 mois de stage au sein de l'équipe RMod de Inria Nord-Europe. Le stage consiste a analyser les différentes méthodes pour transformer le code ESOPE/Fortran77 sous une forme plus facilement manipulable, c'est-à-dire sous la forme d'un *AST*[1].

Mon travail s'est porté sur deux phases :

— trouver un analyseur pour Fortran77 et ESOPE[2] ;

— charger dans l'environnement Moose l'*AST* obtenu à partir du code transformé pour construire un modèle à partir d'un méta-modèle réalisé dans cet environnement.

J'explique la démarche qui à permis de choisir deux analyseurs syntaxiques ainsi que la construction d'un méta-modèle pour *Fortran77*.

Abstract

This report retraces my 5 months internship in the RMod team of Inria Nord-Europe. The internship consists in analysing the different methods to transform the ESOPE/Fortran77 code into a more easily manipulated form, *i.e.* in the form of an AST. My work focused on two phases :

— finding a parser for Fortran77 and ESOPE ;

— loading into the Moose environment the *AST* obtained from the transformed code to build a model from a metamodel made in this environment.

I explain the approach that allowed us to choose two syntactic parsers and the construction of a metamodel for *Fortran77*.

1. Utilisé comme la représentation intermédiaire interne d'un programme informatique
2. ESOPE est une extension du langage fortran 77

Table des matières

Table des figures

Liste des tableaux

1 Remerciements

Je tiens avant tout à remercier toute l'équipe RMod pour son acceuil chaleureux, son soutien tout au long de mon stage et les diverses connaissances que ses membres ont partagé avec moi durant cette période.

J'aimerais ensuite remercier tout particulièrement M. Nicolas Anquetil ainsi que M. Stéphane Ducasse pour leur disponnibilité et leurs précieux conseils.

Je remercie M. Frédéric Hoogstoel pour les retours détaillés qu'il m'a fait sur mes différents rapports pour pouvoir les améliorer. J'ai particulièrement apprécié, lors de sa visite, les explications sur l'intégration de la fiche de compétence dans le rapport.

Je remercie également M. Vincent Aranega qui a également été très disponible pour répondre aux différentes questions que je me posais, sur la méta-modélisation ou la présentation de ANTLR.

Je remercie aussi Clotilde Toullec et Maximilian W. Willembrinck pour les nombreuses réponses qu'ils m'ont apportées notamment dans les phases de déboguages.

Je remercie Santiago Bratignolo et Mahugnon H. Houekpetodji pour l'éclairage qu'ils m'ont apporté sur la poursuite de ce projet en thèse.

Je tiens à remercier les enseignants du département Informatique et Statistique m'ayant permis de suivre une formation d'ingénieur de qualité qui m'a donné les connaissances et compétences nécessaires à la bonne réalisation de ce stage : qu'il s'agisse des cours de langages et traducteurs ou les cours d'ingénierie logicielle. Ces connaissances m'ont permis d'être plus performant lors de mon stage et m'ont donné des outils pour trouver des solutions adaptées aux diverses situations rencontrées.

Enfin je remercie toute l'équipe Framatome pour son accueil lors de ma visite dans ses locaux à Paris, son retour sur ma simulation de soutenance et les échanges que nous avons eu pour l'améliorer.

2 Introduction

Mon stage de fin d'études, d'une durée de 5 mois, réalisé dans le cadre de ma formation d'ingénieur en Informatique et Statistique à Polytech'Lille, se déroule au sein de l'équipe RMod du Centre Inria de l'Université de Lille. Il s'agit d'un stage préparatoire à une thèse portant sur la migration de code source Fortran77/ESOPE vers du Fortran moderne avec un paradigme orienté objet. Durant ce stage, mon travail a consisté à étudier des analyseurs syntaxiques et à identifier parmi eux ceux qui permettent d'analyser à la fois du code Fortran77 et sa sur-couche ESOPE. Ensuite j'ai importé les Arbres de Syntaxe Abstraite - *Abstract Syntax Trees* (AST) correspondants dans la plateforme d'analyse Moose [3].

Dans ce rapport, après vous avoir présenté la structure d'accueil et le contexte dans lequel s'est déroulé ce stage, je vous exposerai ma démarche pour la sélection et la validation d'un choix de solution pour un analyseur syntaxique. Je présente aussi la construction d'un méta-modèle pour `Fortran77` pour l'environnement Moose.

3. Environnement d'analyse qui fournit une panoplie d'outils d'analyses

3 Présentation de la structure d'accueil

3.1 Inria

J'effectue mon stage de fin d'études au sein de l'équipe RMOD de Inria.

Inria est un établissement public à caractère scientifique et technologique français spécialisé en mathématiques et informatique. Inria travaille aussi dans les sciences appliquées aux Technologies de l'Information et de la Communication. L'institut fournit également un transfert de technologie fort et porte une attention particulière pour la formation par la recherche, la diffusion du développement scientifique et technique, l'expertise et la participation à des programmes internationaux.

Le centre Inria de l'université de Lille, créé en 2008, est implanté sur deux sites : à la Haute-Borne, où j'effectue mon stage, au coeur du campus de l'université de Lille et à Euratechnologies. Il compte 15 équipes-projets mobilisant plus de 360 personnes, scientifiques et personnels d'appui à la recherche et à l'innovation, issues d'une quarantaine de nationalités.

RMod [4] est une équipe qui fait de l'analyse et de la construction de langage pour l'évolution d'applications orientées objets. L'objectif de RMod est d'aider à maintenir actives et opérationnelles des applications.

Cet objectif se concrétise suivant deux axes complémentaires :

— la ré-ingénierie ;

— la définition de nouveaux constructeurs dans les langages de programmation.

Dans le cadre de la ré-ingénierie, ils proposent de nouvelles analyses pour comprendre et restructurer de grandes applications (métriques spécialisées, visualisations adaptées).

Dans le contexte des constructeurs pour la modularité, ils travaillent à la validation du modèle de traits [5] ainsi que de nouveaux systèmes de modules et enfin, à la définition d'un noyau sécurisé pour le langage Pharo [6] et son environnement.

3.2 Framatome

Framatome est un leader international de l'énergie nucléaire, reconnu pour ses solutions innovantes et ses technologies à forte valeur ajoutée à destination du parc nucléaire mondial. Forte d'une expertise mondiale et de solides références en termes de fiabilité et de performances, l'entreprise conçoit, entretient et installe des composants et des combustibles ainsi que des systèmes de contrôle-commande pour les centrales nucléaires.

4. Site web de l'équipe.

5. Les traits sont un outil flexible pour éviter les problèmes d'héritages multiples. C'est comme les interfaces dans d'autres langages de programmation comme Java.

6. Langage de programmation largement inspiré de Smalltalk et un IDE.

Ses quelques 15000 collaborateurs permettent chaque jour aux clients de Framatome de fournir un mix énergétique bas-carbone toujours plus propre, plus sûr et plus économique.

Framatome est détenue par le Groupe EDF, Mitsubishi Heavy Industries et Assystem.

4 Contexte général

Ce stage de 5 mois s'est déroulé en plusieurs phases.

— une phase de découverte de Fortran ;
— une phase de recherche bibliographique ;
— une phase d'identification d'un analyseur syntaxique parmi plusieurs étudiés ;
— une phase de traitement de la représentation intermédiaire obtenue (AST) à l'issue de la phase d'analyse dans la plateforme Moose.

Le projet s'inscrit dans le cadre d'une collaboration entre l'équipe RMod et Framatome, société qui gère les logiciels utilisés pour la programmation des centrales nucléaires. Les ingénieurs de Framatome écrivent des simulations dans une sur-couche du langage de programmation Fortran appelée ESOPE. Ils ont ainsi développé une grande base de programmes avec cette sur-couche et souhaitent la faire évoluer vers du Fortran moderne pur, langage utilisé pour les calculs scientifiques. En effet, ils sont contraints avec l'évolution des compilateurs d'enlever une partie des optimisations de leurs codes. A terme, cette situation peut conduire à une impossibilité totale de les compiler.

La raison de cette volonté de migration vient de plusieurs facteurs :

— `Fortran77` est un langage vieillissant qui rend difficile des évolutions des applications écrites dans ce langage ;
— raréfaction des développeurs : très peu d'écoles ou d'universités proposent des formations dans ce langage ;
— une application qui n'évolue pas perd progressivement en qualité ;
— à long terme, les changements deviennent très risqués.

Le stage est un travail préliminaire à un projet de recherche dont l'objectif final est de proposer une solution de traduction automatique de sources ESOPE vers du Fortran moderne permettant une approche orientée objet.

ESOPE est une extension du langage Fortran77. L'objectif d'ESOPE est de faciliter la gestion des données et de permettre la notion d'objet par la structuration des données à l'image des structures en `C`. La motivation était de disposer d'un ensemble de données au sein d'une seule variable. Cette notion était absente du Fortran77. C'est ainsi qu'une entité appelée `SEGMENT` a été ajoutée comme sur la figure 2, ainsi que les primitives permettant de la manipuler. Un `SEGMENT` :

— est un regroupement de variables fortran défini par le programmeur ;
— est référencé par une seule variable appelée POINTEUR. La connaissance du pointeur suffit pour accéder à toutes les variables contenues dans la structure

de données.

```
segment,list
  integer ilist
  integer elist
endsegment
```

Figure 2 – Exemple de définition d'un segment de nom `list`. Il comporte deux champs (`ilist` et `elist`) de type entier.

Il a déjà été démontré, lors d'un précédent stage, qu'il est possible d'effectuer cette traduction de sources ESOPE vers du Fortran2003 manuellement. La quantité importante des sources à traduire ainsi que leurs tailles ne permettent toutefois pas d'envisager une traduction manuelle, ce qui explique le besoin d'une approche automatique.

L'objectif est donc de proposer des solutions de traduction automatique des codes de la sur-couche ESOPE vers du Fortran moderne. En ce sens le stage va consister à analyser les différentes pistes possibles pour transformer du code Fortran77/ESOPE dans une représentation intermédiaire sous forme d'AST. Il sera ainsi possible de transformer cet AST pour qu'il corresponde au langage cible vers lequel on souhaite aller.

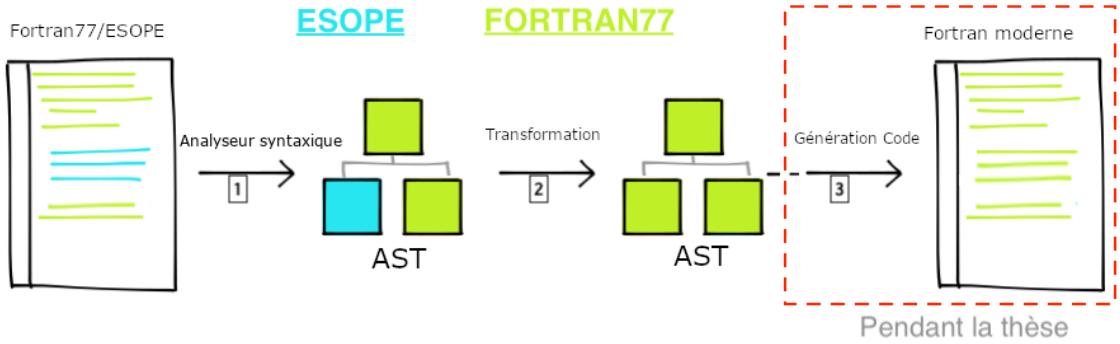


Figure 3 – Les étapes de la migration

La figure 3 présente les étapes d'une approche pour cette migration :

1. Code source en Fortran et sa sur-couche --> AST Fortran+sur-couche : un analyseur syntaxique va prendre en entrée le code source et produire l'AST correspondant ;
2. AST Fortran+sur-couche --> AST Fortran pur : cet AST va être transformé pour remplacer la sur-couche ESOPE par des nœuds correspondants à du Fortran pur. Ainsi chaque instruction ESOPE devient une « annotation » sous forme de commentaire Fortran77 permettant de la reconnaître. Cette

technique de mise en commentaire n'est utilisée que pour les définitions des segments, les instructions de manipulation et d'utilisation d'un segment ainsi que la définition d'un pointeur référençant ce segment. Les autres types d'instructions sont transformées de manière à permettre une analyse syntaxique correcte [7] ;

3. AST Fortran pur `-->` code source Fortran pur : pour finir, une génération de code à partir de l'AST Fortran pur vers du code source Fortran pur. Pour atteindre cet objectif, une première analyse comme une inférence de type simple pourra être faite.

Les AST sont des structures de données importantes dans les frontaux [8] des compilateurs. Ils sont utilisés comme la représentation intermédiaire interne d'un programme informatique à l'issue de la phase d'analyse syntaxique.

Dans le cadre de ce stage, seul les points (`1`) et le chargement de l'AST annoté dans la plateforme Moose (`2`) seront abordés.

Pour le projet complet, plusieurs choses sont nécessaires :

— identifier et comprendre les instructions de la sur-couche ESOPE ;
— identifier un analyseur syntaxique permettant de produire l'AST Fortran avec la sur-couche ESOPE.
— trouver les règles de transformation de cet AST en un AST où la sur-couche a été remplacée par son équivalent Fortran ;
— enfin, reconstruire le code source en Fortran, en partant de l'AST qui a été obtenu à l'étape précédente.

Lors de ce stage, les deux premières étapes ont été réalisées. Les suivantes seront l'objet d'un projet de thèse.

Pour réaliser ces objectifs, il est nécessaire d'identifier un analyseur syntaxique capable de produire une représentation intermédiaire sous forme d'AST à partir de sources Fortran77 et ESOPE. Cet analyseur syntaxique devra être à code ouvert pour permettre une modification en intégrant les règles de traduction de la sur-couche ESOPE contenue dans le code source initial ou de pouvoir modifier le format de sortie.

La représentation du code source se fera sous forme d'AST. Cette étape se fera par une succession d'étapes combinant des analyses lexicales et syntaxiques. Cet analyseur syntaxique doit donc être capable de prendre en entrée le code source initial et de générer, en utilisant les règles syntaxiques du langage source, un arbre syntaxique.

7. Les détails de cette traduction seront évoqués plus loin dans ce rapport
8. C'est la partie du compilateur qui prend en charge les analyses lexicale, syntaxique et sémantique

5 Travail réalisé

5.1 Moyens mis en oeuvre

J'ai commencé par la découverte de la syntaxe de base de Fortran77. J'ai aussi appris la représentation de code source à travers des AST ainsi que la façon de les parcourir avec des visiteurs [9]. J'ai également étudié un analyseur syntaxique OFP écrit en java qui permet de prendre du code Fortran90 et 2003 en entrée et de le transformer en AST au format XML. J'ai focalisé mon apprentissage sur la manipulation d'AST avec le langage Pharo. J'ai ainsi pu générer des classes pour chaque expression ou instruction Fortran correspondant aux noeuds de l'arbre. Une fois l'AST chargé dans pharo, j'ai également pu le parcourir en utilisant des algorithmes de parcours d'arbre notamment le parcours en profondeur et la manipulation des noeuds de l'arbre en utilisant le patron de conception visiteur.

5.2 Recherche bibliographique

Les travaux de `Ralf W. Grosse-Kunstleve et al.` [Gro+12] sur FABLE, application écrite en python qui permet de traduire du code Fortran vers du C++, m'ont permis de comprendre que l'enjeu majeur de la migration automatique est le temps de travail manuel nécessaire pour arriver à un code fonctionnel et lisible dans le code du langage cible.

L'article `A prototype of Fortran-to-Java converter` [Fox+97] traite de la conversion de code `Fortran` vers du `Java`. Il traite le portage des bibliothèques Fortran vers des langages plus modernes comme le `C` et `Java`, qui étaient considérés comme lents à cette période et sont maintenant beaucoup plus rapides. Les auteurs voulaient porter les anciennes bibliothèques Fortran vers Java afin de profiter de la portabilité et de la popularité de Java. Mais ils se sont heurtés à de nombreux problèmes qui les ont conduits à des implémentations de manière procédurale.

Les deux articles cités précédemment([Fox+97] et [Gro+12]) parlent de `f2c` [10] [Fel90] dont ils se sont inspirés. Il répond au manque de spécialistes du Fortran. Il permet de faire évoluer les bibliothèques de calculs numériques Fortran avec des programmes `C` plus récents. Il est à la base de beaucoup d'autres traducteurs ou compilateurs `Fortran` actuels. Le code obtenu est cependant difficile à lire et à maintenir.

`L. Moonen` explique le fonctionnement des « *island grammars* » qui décrivent une partie d'un langage sous-jacent. J'ai utilisé cette méthode pour identifier et annoter les instructions ESOPE. En utilisant la métaphore d'une île entourée d'eau : les îlots correspondent aux instructions ESOPE et l'eau au reste des instructions.

9. https://github.com/UnivLille-Meta/l3s6-meta-ressources/tree/session-2022/Support
10. Cet article explique la conversion de Fortran77 vers du code en C.

5.3 Identification des analyseurs syntaxiques Fortran77/ESOPE

L'objectif de cette phase est de comparer un certain nombre d'analyseurs syntaxiques sur les critères définis, pour identifier un ou deux « bons » candidats capables de transformer à la fois du code ESOPE et du code Fortran77 sous la forme d'un AST. Un analyseur syntaxique doit être capable d'analyser :

- du Fortran77 idiomatique [11] ;
- du Fortran77 généré par ESOPE [12] ;

Il est souhaitable, en plus de ces critères requis, que chacun des analyseurs syntaxiques puisse offrir les avantages suivants :

- être capable d'offrir en sortie une représentation intermédiaire sous forme d'AST. Cet AST doit être dans un format facile à utiliser (en XML par exemple) ;
- garder les commentaires du code source analysé ; cela pour deux raisons essentielles :
  - le code source de base contient des commentaires, il est nécessaire de ne pas les supprimer ;
  - les commentaires vont être utilisés pour les instructions `ESOPE` ;
- donner accès aux positions dans le fichier source des noeuds de l'AST. Cela permet de repérer la position d'une instruction dans le source original ; utile dans la plateforme Moose pour interroger le modèle construit par exemple.
- offrir une table des symboles ;
- être à source ouverte de façon à être adaptable aux besoins du projet.

J'ai identifié les analyseurs présentés dans les sections suivantes, que j'ai testés et confrontés aux critères définis.

5.3.1 Les analyseurs Fortran77

5.3.1.1 gfortran

`gfortran` est un compilateur fortran, un frontal Fortran inclus dans la suite `gcc`. Il implémente complètement les standards Fortran77, 90, 95 et une grande majorité des standards des fortran modernes jusqu'à la version 2018. L'intérêt pour ce compilateur réside donc dans le fait d'être capable de produire une représentation interne sous forme d'AST après la phase d'analyse syntaxique. Il permet d'analyser du Fortran77 idiomatique mais également du Fortran77 généré par ESOPE. Pour cela il faut préciser qu'on souhaite analyser du fortran dans un dialecte suivant le standard 77 "`legacy`" avec la commande suivante :

```
gfortran -std=legacy -I path/to/includedir \
    -c fortran-source.f -fdump-fortran-original > output.ast
```

11. On parle de code idiomatique lorsqu'on s'efforce de respecter la philosophie (la manière de faire) du langage de programmation dans lequel il est écrit.
12. Cette contrainte est optionnelle.

On obtient alors la table des symboles ainsi que l'AST de la figure 4. Toutefois, l'AST produit est dans un format « non standard ». Il nécessite donc un traitement qui peut se faire en créant une grammaire pour le format de sortie.

En revanche, il ne permet pas :

— de garder les commentaires ;

— de garder les positions des instructions dans le fichier source ;

```
    code:
    ASSIGN stgbrk:status 0
    ASSIGN stgbrk:nlist 10
    CALL ooowin ((stgbrk:oo4) (0) ('STGBRK 20 RLIST ') (stgbrk:oo1) ((+ 2 stgbrk:nlist)))
    ASSIGN stgbrk:oo_001((+ (+ -2_8 stgbrk:oow((+ stgbrk:oot __convert_i4_i8[[((stgbrk:oo1))]]))) 2_8)) stgbrk:nlist
    ASSIGN stgbrk:rlist stgbrk:oo1
    ASSIGN stgbrk:oo1 stgbrk:rlist
    CALL ooowac ((stgbrk:oo4) (0) ('STGBRK 24 RLIST ') (stgbrk:oo1) (0))
    ASSIGN stgbrk:oo_002((+ (+ -4_8 stgbrk:oow((+ stgbrk:oot __convert_i4_i8[[((stgbrk:rlist))]]))) 1_8)) 0
    ASSIGN stgbrk:oo1 stgbrk:splist
    CALL ooowac ((stgbrk:oo4) (0) ('STGBRK 27 SPLIST ') (stgbrk:oo1) (0))
    ASSIGN stgbrk:oo1 stgbrk:alist
    CALL ooowac ((stgbrk:oo4) (0) ('STGBRK 29 ALIST ') (stgbrk:oo1) (0))
    DO  99001 stgbrk:ia=1 stgbrk:oo_003((+ (+ -6_8 stgbrk:oow((+ stgbrk:oot __convert_i4_i8[[((stgbrk:alist))]]))) 1_8)) 1
      ASSIGN stgbrk:xs __abs_i4[[((stgbrk:oo_004((+ (+ (+ -8_8 stgbrk:oow((+ stgbrk:oot __convert_i4_i8[[((stgbrk:alist))]]))) 2_8) __convert_i4_i8[[((stgbrk:ia))]]))))]]
      ASSIGN stgbrk:oo1 stgbrk:xs
      CALL ooowac ((stgbrk:oo4) (0) ('STGBRK 33 XS ') (stgbrk:oo1) (0))
      ASSIGN stgbrk:lxs stgbrk:oo_005((+ (+ -10_8 stgbrk:oow((+ stgbrk:oot __convert_i4_i8[[((stgbrk:xs))]]))) 1_8))
      IF (== stgbrk:lxs 0)
      ELSE
        ASSIGN stgbrk:first 1
        ASSIGN stgbrk:last stgbrk:lxs
100     CONTINUE
110     ASSIGN stgbrk:i 0
        ASSIGN stgbrk:lmin -1
120     ASSIGN stgbrk:i (+ stgbrk:i 1)
        IF (<= stgbrk:i (parens stgbrk:oo_006((+ (+ -12_8 stgbrk:oow((+ stgbrk:oot __convert_i4_i8[[((stgbrk:splist))]]))) 1_8))))
          ASSIGN stgbrk:sp __abs_i4[[((stgbrk:oo_007((+ (+ (+ -14_8 stgbrk:oow((+ stgbrk:oot __convert_i4_i8[[((stgbrk:splist))]]))) 2_8) __convert_i4_i8[[((stgbrk:i))]]))))]]
          ASSIGN stgbrk:oo1 stgbrk:sp
          CALL ooowac ((stgbrk:oo4) (0) ('STGBRK 46 SP ') (stgbrk:oo1) (0))
          ASSIGN stgbrk:lsp stgbrk:oo_008((+ (+ -16_8 stgbrk:oow((+ stgbrk:oot __convert_i4_i8[[((stgbrk:sp))]]))) 1_8))
          IF (== stgbrk:lsp 0)
            ASSIGN stgbrk:idx 1
          ELSE
            ASSIGN stgbrk:idx __index_1_i4[[((stgbrk:oo_009((+ (+ -18_8 stgbrk:oow((+ stgbrk:oot __convert_i4_i8[[((stgbrk:xs))]]))) 1_8))((+ (+ stgbrk:oov(2) 4) stgbrk:first):(+ (+ stgbrk:oov(2)
            4) stgbrk:last))) (stgbrk:oo_010((+ (+ -20_8 stgbrk:oow((+ stgbrk:oot __convert_i4_i8[[((stgbrk:sp))]]))) 1_8))((+ (+ stgbrk:oov(2) 4) 1):(+ (+ stgbrk:oov(2) 4) stgbrk:oov((+
            stgbrk:oow((+ stgbrk:oot __convert_i4_i8[[((stgbrk:sp))]]))) 1_8)))) ((arg not-present)) ((arg not-present))]]
          ENDIF
          IF (== stgbrk:idx 1)
            IF (== stgbrk:lmin -1)
              ASSIGN stgbrk:lmin stgbrk:lsp
            ENDIF
            IF (== stgbrk:lmin 0)
              ASSIGN stgbrk:lmin stgbrk:lsp
            ENDIF
            IF (AND (parens (< stgbrk:lsp stgbrk:lmin)) (parens (> stgbrk:lsp 0)))
              ASSIGN stgbrk:lmin stgbrk:lsp
            ENDIF
          ENDIF
          ASSIGN stgbrk:oo1 stgbrk:sp
          CALL ooowde ((stgbrk:oo4) (0) ('STGBRK 62 SP ') (stgbrk:oo1) (1))
          GOTO 120
        ENDIF
```

Figure 4 – AST obtenu avec gfortran

5.3.1.2 Fortran-src

`fortran-src`[13], écrit en Haskell, est un frontal de CamFort. CamFort, projet de recherche développé par les Universités de Cambridge et Kent, est un outil de ré-usinage et de vérification pour des programmes Fortran scientifiques. Il supporte les standards de Fortran 66, 77, 90, 95 et partiellement 2003.

Fortran-src supporte ainsi les mêmes standards que CamFort. Il fournit des analyses lexicales, syntaxiques et statiques précoces du code Fortran. Lors de l'analyse syntaxique, on peut spécifier le standard souhaité de la manière suivante :

```
fortran-src -I path/to/includedir source.f > output.ast
```

Cette commande permet d'obtenir un AST comme celui de la figure 5 mais sans la table des symboles dans un format Haskell Algebraic data type (ADT)[14]. Pour

13. github du projet
14. obtenir plus d'information sur la description de ce format

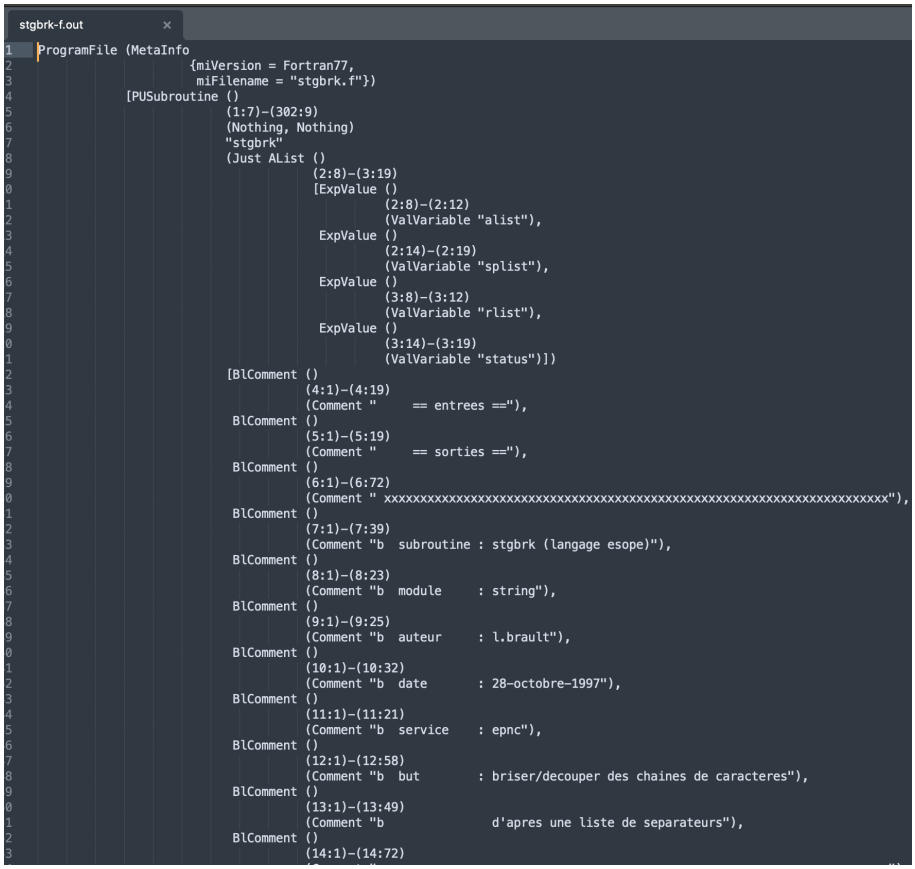
être traité, ce format `ADT` nécessite également une analyse pour obtenir un format facile à utiliser. Cela peut se faire soit en modifiant le code source `Haskell` pour obtenir le format souhaité ; soit en écrivant une grammaire capable d'analyser ce format. J'ai écrit en partie une grammaire pour analyser la sortie en PetitParser2 [15] et en ANTLR.

Fortran-src permet de :

— garder les commentaires ;

— garder les positions des instructions du fichier source.

Il n'existait pas de fonctionnalité permettant d'obtenir l'AST au format souhaité (XML ou JSON). J'ai entrepris de la développer. Devant l'ampleur de la tâche, j'ai échangé [16] avec les contributeurs du projet qui travaillent actuellement au développement d'une version offrant cette fonctionnalité. J'ai pu la tester et obtenir des sorties dans des formats qui seront plus faciles à manipuler. Cette piste ne sera pas exploitée dans le cadre de ce stage compte tenu de son arrivée tardive mais sera un atout pour la suite.



```
stgbrk-f.out
ProgramFile (MetaInfo
                    {miVersion = Fortran77,
                     miFilename = "stgbrk.f"})
            [PUSubroutine ()
                          (1:7)-(302:9)
                          (Nothing, Nothing)
                          "stgbrk"
                          (Just AList ()
                                      (2:8)-(3:19)
                                      [ExpValue ()
                                                (2:8)-(2:12)
                                                (ValVariable "alist"),
                                       ExpValue ()
                                                (2:14)-(2:19)
                                                (ValVariable "splist"),
                                       ExpValue ()
                                                (3:8)-(3:12)
                                                (ValVariable "rlist"),
                                       ExpValue ()
                                                (3:14)-(3:19)
                                                (ValVariable "status")])
                          [BlComment ()
                                     (4:1)-(4:19)
                                     (Comment "     == entrees =="),
                           BlComment ()
                                     (5:1)-(5:19)
                                     (Comment "     == sorties =="),
                           BlComment ()
                                     (6:1)-(6:72)
                                     (Comment " xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"),
                           BlComment ()
                                     (7:1)-(7:39)
                                     (Comment "b  subroutine : stgbrk (langage esope)"),
                           BlComment ()
                                     (8:1)-(8:23)
                                     (Comment "b  module     : string"),
                           BlComment ()
                                     (9:1)-(9:25)
                                     (Comment "b  auteur     : l.brault"),
                           BlComment ()
                                     (10:1)-(10:32)
                                     (Comment "b  date       : 28-octobre-1997"),
                           BlComment ()
                                     (11:1)-(11:21)
                                     (Comment "b  service    : epnc"),
                           BlComment ()
                                     (12:1)-(12:58)
                                     (Comment "b  but        : briser/decouper des chaines de caracteres"),
                           BlComment ()
                                     (13:1)-(13:49)
                                     (Comment "b              d'apres une liste de separateurs"),
                           BlComment ()
                                     (14:1)-(14:72)
```

Figure 5 – AST obtenu avec fortran-src

15. Sera présenté plus loin en section `5.3.2.1`
16. https://github.com/camfort/fortran-src/issues/215

5.3.1.3 OFP

Open Fortran Parser (OFP) fournit un analyseur syntaxique pour Fortran. Cet analyseur est basé sur une grammaire ANother Tool for Language Recognition (ANTLR). Il utilise les outils fournis par ANTLR pour générer l'analyseur Fortran, écrit en Java. Pendant les phases de tests et d'apprentissage de Fortran, au mois de février, j'ai essentiellement utilisé une extension à OFP qui permet d'obtenir un AST en format XML.

Son inconvénient majeur est de ne pas prendre en charge le Fortran77. Par ailleurs l'AST produit à partir des tests effectués sur des sources en Fortran90 n'est pas fiable également. Je m'en suis rendu compte lors du traitement de l'AST dans Pharo. En effet le visiteur qui permet de traiter la classe correspondant au noeud d'une variable contenait des noms de variables vides comme sur la figure 6 :

```
<variables count="2">
  <variable hasInitialValue="false" name="N">
    <entity-decl hasArraySpec="false" hasCharLength="false" hasCoarraySpec="false" hasInitialization="false" id="N" rule="503008"/
  </variable>
  <variable hasInitialValue="false" name="i">
    <entity-decl hasArraySpec="false" hasCharLength="false" hasCoarraySpec="false" hasInitialization="false" id="i" rule="503008"/
  </variable>
  <variable/>
</variables>
```

L'erreur vient du Parseur

Figure 6 – Exemple de noeud ast avec une erreur sur des variables vides

L'intérêt pour cet analyseur syntaxique résidait dans le fichier de sortie au format XML qui serait donc facile à traiter. En effet, des outils existent dans l'environnement Pharo pour charger du XML. Il serait ainsi facile à partir de l'existant de créer ou générer des classes correspondant aux noeuds de l'arbre et de définir des visiteurs pour chacune des classes.

5.3.1.4 f2c

`f2c` [17] est un traducteur de code Fortran77 vers du code `C`. Il est écrit en `C`. Il est utilisé dans beaucoup d'outils qui font de la migration ou des compilateurs comme `gfortran`. Lui même et le code `C` qu'il produit sont donc facilement compilables sous les systèmes de types `Unix`.

La documentation indique qu'il passe par un arbre d'analyse avant de faire la conversion. Pourtant je n'ai trouvé aucun moyen d'exporter cet arbre.

Une solution envisageable serait de faire une traduction directe du code `Fortran77/ESOPE` vers du code (`Fortran2003`) sans avoir à passer par un AST, comme indiqué dans l'article [Fel90].

D'après cet autre article [Lev95] il existe un outil permettant d'améliorer la lisibilité du code source obtenu par `f2c`. Malheureusement le lien hébergeant l'outil n'est plus accessible [18].

17. Le code source de cet outil est disponible ici
18. Voir sur webarchive

5.3.1.5 lfortran

`lfortran` est un compilateur Fortran interactif. Il est construit au-dessus de `LLVM` comme un frontal capable d'analyser du code Fortran moderne. L'étude de cet analyseur syntaxique permet de comprendre l'approche par laquelle il analyse du Fortran. L'idée étant de voir s'il est possible de l'étendre pour une prise en charge du standard Fortran77.

— il ne permet pas d'analyser du Fortran77 ;
— les tests effectués sur du Fortran90 et suivants, permettent de montrer qu'il conserve les commentaires.

Le code source étant disponible, il pourrait être intéressant de voir s'il est possible d'ajouter ou de faire une demande de prise en charge de Fortran77. Cela semble requérir un travail important. Le projet est jeune et peut évoluer. Il reste intéressant de suivre son évolution pour voir si, dans le futur, il est prévu une prise en charge des anciens dialectes Fortran.

5.3.1.6 ANTLR

ANother Tool for Language Recognition (ANTLR) est un générateur de frontaux pour compilateurs qui à partir d'une grammaire en Backus–Naur forme (BNF) [19] va produire des analyseurs lexicaux et syntaxiques.

Pour fonctionner ANTLR a besoin d'une grammaire pour l'analyseur lexicale (sous forme d'expression régulière et d'un automate fini déterministe) et une autre pour l'analyseur syntaxique (en BNF et d'un automate à pile).

Les tests [20] effectués avec cet outil à partir d'une grammaire `Fortran77` [21] permettent d'analyser du code source de ce langage. Mais cette grammaire ne permet pas d'analyser le code `Fortran77` généré par `ESOPE`. En effet `ESOPE` n'impose pas de séparer les « *mots* ». Ce qui finit par faire échouer l'analyseur lexical. Par exemple « `INTEGERVAR` » devrait être analysé en « `INTEGER` » et « `VAR` ». La solution serait donc de trouver un moyen de modifier l'expression régulière, spécifiée dans une grammaire propre à l'analyseur lexical, pour palier à ce type de problèmes. Les tentatives de modifications de l'analyseur lexical n'ont pas été concluantes.

5.3.1.7 BNFC

BNFC est aussi un constructeur de frontal de compilateur. Il permet, à partir d'une grammaire en BNF labellisée, de produire un ensemble d'outils :

— un analyseur lexical ;
— un analyseur syntaxique ;
— un analyseur ANTLR.

19. https://github.com/antlr/grammars-v4/tree/master/fortran77
20. Un compte rendu de ce retour d'expérience est disponible ici.
21. A partir de cette grammaire.

La grammaire n'étant pas disponible, il faudra donc créer une grammaire pour le `Fortran77` mais aussi le `Fortran77` généré avec `ESOPE`.

J'avais commencé à créer une grammaire en m'inspirant de celle-ci [22], mais elle ne parvient pas à analyser du Fortran77 généré par ESOPE.

L'écriture d'une grammaire étant complexe et demandant beaucoup de temps, cette solution n'a pas pu être poursuivie.

5.3.1.8 ROSE

ROSE est un framework de compilation à source ouverte. C'est une infrastructure pour des compilateurs permettant de générer des analyseurs et des traducteurs de source-à-source pour de multiples langages, notamment C, C++ et Fortran. Contrairement à la plupart des autres compilateurs de recherche, ROSE vise à permettre aux non experts de tirer partie des technologies de compilation pour construire leurs propres analyseurs et optimiseurs de logiciels personnalisés.

La procédure d'installation décrite ici [23], n'ayant pas fonctionné sur ma machine sous macOS, je n'ai pu le tester.

5.3.1.9 flang

`flang` [24] est un frontal de LLVM [25].

Il compile du Fortran77 mais ne permet pas d'obtenir l'AST. Il a un comportement étrange quand on demande à obtenir l'AST sur un Fortran77. Il modifie le fichier source en écrasant son contenu.

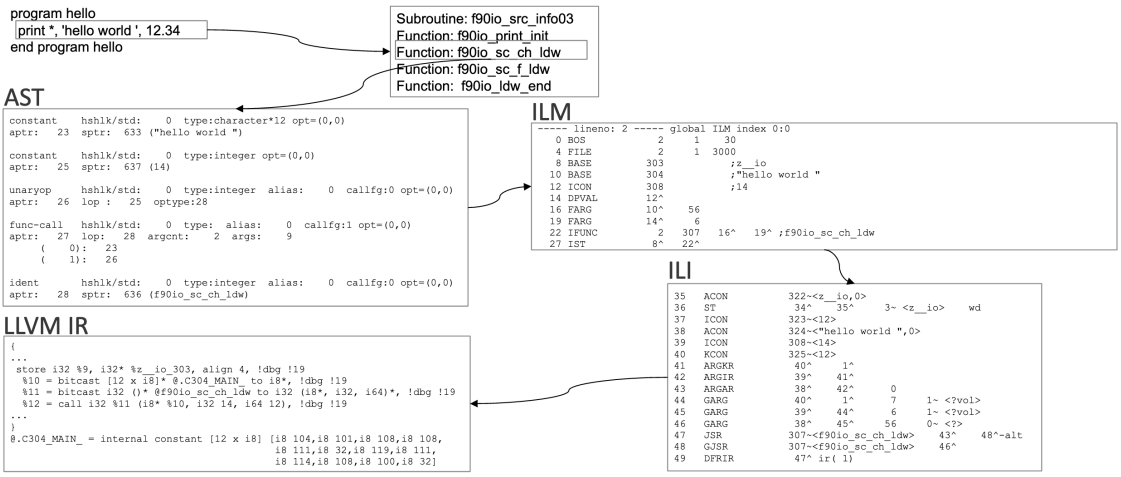


Figure 7 – AST avec flang pour du fortran90

22. https://slebok.github.io/zoo/fortran/f90/waite-cordy/extracted/index.html
23. https://github.com/rose-compiler/rose/wiki/Install-Rose-From-Source
24. Le site web du projet.
25. Flang llvm frontend.

Le test que j'ai effectué sur un exemple simple sur du Fortran90 permet d'obtenir une structure comme celle sur la figure 7.

5.3.1.10 iFortran

`iFortran` est le compilateur `Fortran` de `Intel`. Cet analyseur syntaxique est celui qui est utilisé par les ingénieurs de Framatome pour compiler leurs sources. Je n'ai trouvé aucune information sur la possibilité d'obtenir un AST [26]. Ce compilateur étant propriétaire, je n'ai pas pu poursuivre des investigations plus poussées.

5.3.2 Les analyseurs ESOPE

5.3.2.1 PetitParser

`PetitParser2` est un framework pour la construction d'analyseurs syntaxiques. Il est développé par Jan Kurš. Il met l'accent sur la performance et la flexibilité. *`PetitParser2`*, utilise une combinaison unique de quatre méthodologies alternatives d'analyse syntaxique :

- les analyseurs sans scanner (également appelés analyse syntaxique sans lexeur) [27] ;
- les combinateurs d'analyseurs syntaxiques ;
- les grammaires d'expression d'analyse syntaxique ;
- les analyseurs Packrat.

Une « *island grammar* » a été définie en `PetitParser2`. Elle permet de reconnaître les instructions ESOPE contenues dans un fichier source ESOPE. Ainsi toutes les instructions ESOPE reconnues sont mises en commentaire (à l'image des annotations dans les langages de programmation tels que `Java`). Cela permet de les conserver dans les sources et d'être syntaxiquement valides pour un analyseur syntaxique `Fortran77`.

Les règles utilisées pour obtenir les fichiers annotés sont les suivantes :

- les segments (`segact`, `segadj`, `segini`, `segsup`, `segdel`) et les instructions de type `pointeur abc.def` sont annotées avec le suffixe `cESO` sur les 6 premières colonnes de chaque ligne de ces instructions, par exemple : `segadj,aplist` devient `cESO segadj,aplist`.
  A noter que lors de la définition d'un segment, on annote aussi les définitions des champs contenus dans ce segment pour éviter les erreurs d'analyse syntaxique ;
- les utilisations d'un pointeur sur une instruction d'affectation `abc.def=0` ou comme une expression dans une boucle, par exemple : `do i, abc.def` sont traduites par un tableau `ESOat(abc,def)` ;

26. https://community.intel.com/t5/Intel-Fortran-Compiler/Generating-AST-with-ifort/td-p/1317247
27. Effectue la tokenisation (décomposition d'un flux de caractères en mots) et l'analyse syntaxique en une seule étape, plutôt que de la décomposer en un pipeline composé d'un lexeur suivi d'un analyseur syntaxique, exécutés simultanément.

— les instructions de ce type `abc.def(1)` sont traduites comme ceci : `ESOsl(ESOat(abc,def),1)` ;
— les instructions du type `rlist.elist(rlist.ilist)=ms)` sont traduites en : `ESOar(ESOat(rlist,elist),ESOat(rlist,ilist))=ms`.

5.3.2.2 ESOPE

Le code source d'ESOPE est écrit en ESOPE. L'outil qui permet de faire la traduction de l'ESOPE vers du Fortran77 est également appelé ESOPE. Pour passer du code ESOPE vers du code Fortran77, le cheminement est le suivant :

— les directives au préprocesseur (`#include`, `#ifdef`, *etc*) sont traitées par un préprocesseur qui fait des transformations purement textuelles (remplacement de chaînes de caractères, inclusion d'autres fichiers sources, *etc*) de la façon suivante : `source.E -> source.e` ;
— les fichiers « `source.e` » préprocessés sont traduits en source Fortran77 « `source.f` » par le traducteur ESOPE ;
— les fichiers sources Fortran77 « `source.f` » sont compilés avec un compilateur Fortran77.

Cet analyseur est intéressant notamment pour comprendre la façon dont il réalise sa traduction de sources `ESOPE` vers le Fortran généré. J'aurais pu m'inspirer de la grammaire qui est utilisée dans `ESOPE` pour l'adapter soit pour ANTLR ou pour un des analyseurs nécessitant une grammaire comme ceux cités plus haut. La transmission du traducteur étant soumise à des contraintes administratives, je n'ai pas pu mener des expériences avec cet analyseur.

5.4 Choix d'un analyseur syntaxique

Les notations des critères du tableau suivant correspondent à :

1. **F77** : analyse du Fortran77 idiomatique ;
2. **ESP** : analyse du Fortran77 généré par ESOPE ;
3. **AST** : offre un AST en sortie d'analyse ;
4. **FMT** : cet AST est dans un format facile a utiliser (ex typique : dump XML) ;
5. **CMT** : l'analyseur garde les commentaires ;
6. **POS** : donne accès aux positions dans le fichier des noeuds de l'AST ;
7. **TS** : offre une table des symboles ;
8. **OS** : analyseur à source ouverte.

Tableau récapitulatif de l'analyse des parseurs								
Critères / Parseurs	F77	ESP	AST	FMT	CMT	POS	TS	OS
`Gfortran`	Oui	Oui	Oui	Inconnu	Non	Non	Oui	Oui
`Fortran-src`	Oui	Oui	Oui	ADT	Oui	Oui	Oui	Oui
OFP	Non	Non	Oui	XML	Non	Non	Oui	Oui
`f2c`	Oui	Non	Non					
ANTLR	Oui* [28]	Oui*	Oui					
BNFC	Oui*	Oui*	Oui	XML	Non	Non	Oui	Oui
`lfortran`	Non	Non	Oui	Inconnu	Oui	Non	Non	Oui
`flang`	Non	Non						
`rose`	Ne s'installe pas sous MacOS							
`ESOPE`	Oui	Oui						
`PetitParser2`	Non	Oui	Oui		Oui	Oui	Non	Oui
`iFortran`	Oui	Oui						

Table 1 – Tableau récapitulatif des différents analyseurs étudiés

Compte tenu des critères évoqués et des tests effectués sur l'ensemble des sources fournies, on distingue deux analyseurs syntaxiques prometteurs : il s'agit de `gfortran` et `fortran-src`. Cependant ces deux analyseurs nécessitent un traitement des formats des AST. En revanche, ils permettent d'analyser à la fois des codes en `Fortran77` idiomatique et généré `ESOPE`.

Ainsi mon travail s'est concentré sur l'utilisation de `fortran-src`. Le choix de porter l'effort sur ce dernier est essentiellement dû à sa capacité à garder à la fois les commentaires et les positions des instructions. Par ailleurs, comme évoqué plus haut, il y a actuellement un développement en cours pour obtenir une sortie dans un

28. * : Nécessite de fournir une grammaire

format d'échange en JSON. La présence des commentaires après l'analyse syntaxique effectuée par *fortran-src* me permet de réaliser la première phase de génération de sources Fortran77 annotées compatibles avec lui.

6 Mise en oeuvre de la solution retenue

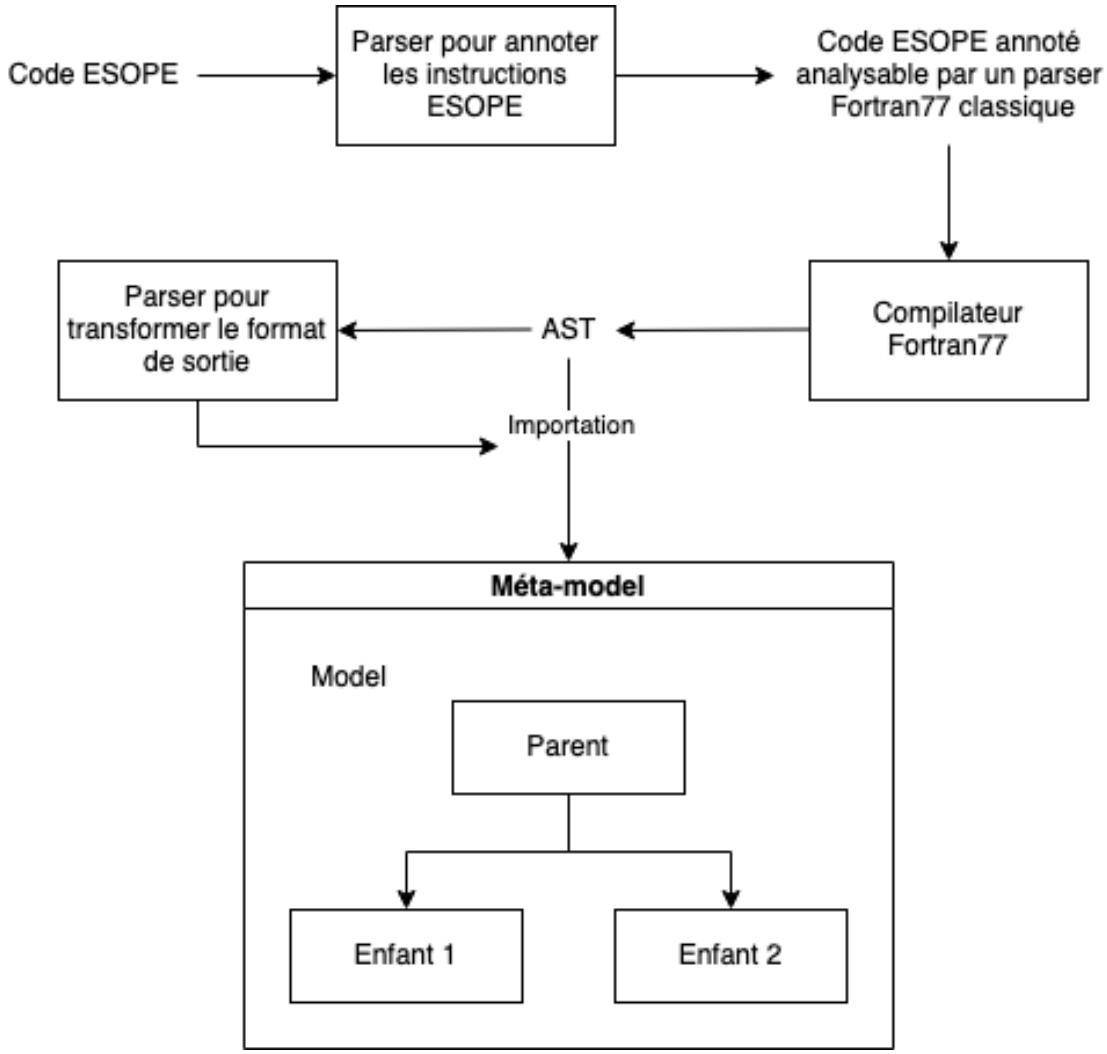


Figure 8 – La solution mise en oeuvre

Dans cette partie je vais présenter la solution mise en oeuvre depuis les sources ESOPE jusqu'à l'obtention d'un meta-modèle Fortran77 comme sur la figure 8.

Je pars des fichiers ESOPE déjà préprocessés, par exemple, dans lesquels les instructions d'inclusions ont été traitées. Il s'agit donc des fichiers avec une extension « `.e` » en minuscule.

L'analyseur construit avec *PetitParser2* va prendre en entrée ces fichiers ESOPE et mettre toutes les instructions ESOPE sous forme d'annotation en utilisant les commentaires Fortran77 (*cESO*). L'objectif étant d'obtenir des fichiers annotés dans lesquels les instructions ESOPE ont été mises en commentaires fortran77 valides. Ces fichiers sont donc du Fortran77 légal même s'ils ne peuvent pas être compilables en l'état. En effet certaines instructions ne sont plus accessibles. Ces fichiers restent toutefois analysables syntaxiquement d'une part et d'autre part, je suis capable de retrouver les instructions ESOPE annotées dans les AST construits par l'analyseur syntaxique *fortran-src*. Les types d'instructions annotées prises en compte par cet analyseur avec ce mécanisme d'annotation sont les suivants :

- les déclarations de type *SEGMENT* comme sur la figure 2 ;
- les déclarations des pointeurs sur les segments (*pointeur*) ;
- les instructions opérant sur les segments (*segini*, *segact*, *segadj*, *segsup*, *segdel*, *segdef*).

Les instructions ESOPE plus complexes, contenues dans des instructions Fortran77, sont traitées différemment pour éviter la perte de la validité syntaxique. Il s'agit des instructions suivantes :

- les accès aux dimensions de la forme *abc.def(/1)* [29] ;
- les instructions d'affectation de champs pour les instructions pointées de la forme *abc.def=variable/value* [30] ;

Pour l'instant, il existe une catégorie d'instructions ESOPE non prise en compte à ce stade. Il s'agit des notations pointées implicites [31] de la forme *def=variable*. Elles sont traitées comme du Fortran77 normal. Je n'ai pas trouvé de moyen de les différencier d'une instruction fortran77. Cela pourra être résolu dans la phase d'analyse dans l'environnement Moose.

Fortran-src va donc construire les AST à partir des fichiers annotés ESOPE.

Une fois ces AST obtenus, j'ai créé le méta-modèle fortran77 dans l'environnement Moose. Ce méta-modèle me permet de modéliser les concepts principaux du langage à savoir les notions de :

- `ProgramFile` : est un fichier contenant du fortran dans lequel on peut retrouver des sous-programmes `ProgramUnit`.
- `ProgramUnit` : est constitué d'un des éléments suivants :
  - `Program` : est le sous-programme équivalent au programme principal. Il peut également y avoir un nombre quelconque de procédures externes (fonctions et sous-routines) et de sous-programmes blockdata.
  - `BlockData` : est un type de sous-programme qui se compose de l'instruction `blockdata` ; de toutes les déclarations de type nécessaire, d'une liste des blocs `COMMON` nommés et de leurs variables, et d'une ou plusieurs instructions DATA initialisant une ou plusieurs des variables apparaissant

29. Traduite en : *ESOat(abc, def, 1)*
30. Traduite en : *ESOat(abc, def, 1)*
31. Ne contenant que le nom du champs sans préfixe par le nom du pointeur sur le segment

dans les blocs `COMMON`. Son seul but est d'initialiser les valeurs des blocs `COMMON` nommés.
  - `Function` : est un type de sous-programme qui retourne sa valeur de retour au travers de son nom.
  - `Subroutine` : pour ce sous-programme, les valeurs de retour se font à travers les arguments de la sous-routine ;
- `Comment` : cette entité va permettre de modéliser les commentaires qui sont des éléments importants dans l'idée de la solution.
- `Variable` : va contenir les accès faits par un sous-programme ;
- `Paramètre` : sont les variables qu'un sous-programme attend en argument.

Dans un premier temps j'ai utilisé le schéma UML de la figure 10 pour construire mon méta-modèle. A partir de ce schéma, j'ai construit mon méta-modèle tel que présenté sur la figure 9.

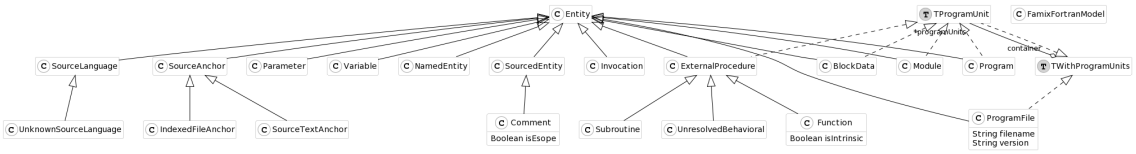


Figure 9 – Méta-modèle fortran construit dans l'environnement Moose

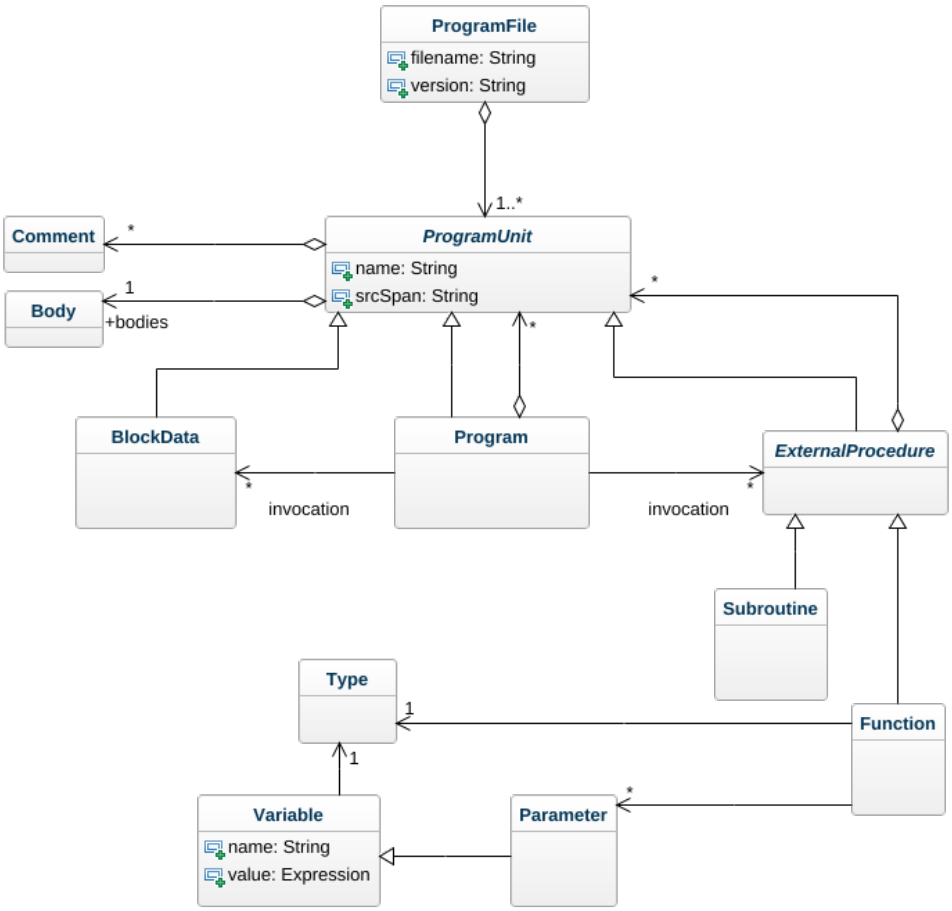


Figure 10 – Diagramme de classes UML du méta-modèle à construire

Après la création du méta-modèle, un deuxième analyseur, écrit en *PetitParser2*, analyse l'AST obtenu avec *fortran-src*. En effet ce dernier produit un fichier de sortie au format ADT d'Haskell. Il a fallu créer une grammaire pour analyser ce fichier. A partir des entités obtenues grâce au méta-modèle, des objets correspondant aux noeuds de l'AST sont créés comme sur la figure 11 où on voit le modèle construit et sur lequel on peut effectuer des opérations pour l'interroger.

On peut voir sur la figure 11 les commentaires se trouvant dans la sous-routine *stgbrk*. Les commentaires correspondant aux annotations sont suffixés par la chaîne de caractère `[ESOPE]`.

On peut également voir sur la figure 12 les invocations d'un sous-programme vers un autre sous-programme. Par exemple sur cette figure, on voit que la sous-routine *string* fait appel à la sous-routine *strlen*. On remarque également des invocations spéciales correspondant aux transformations faites sur des instructions `ESOPE`.

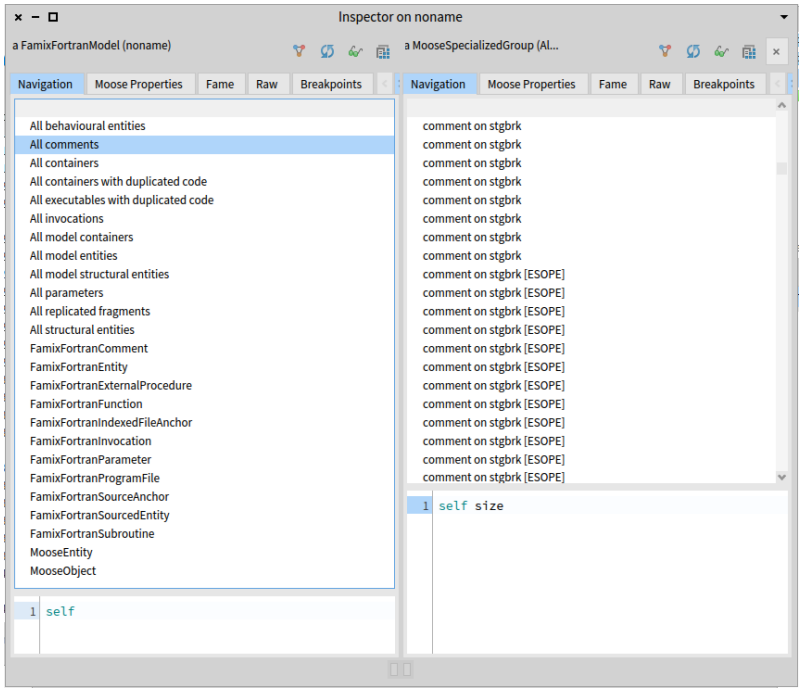


Figure 11 – Les commentaires contenus dans les ASTs chargés

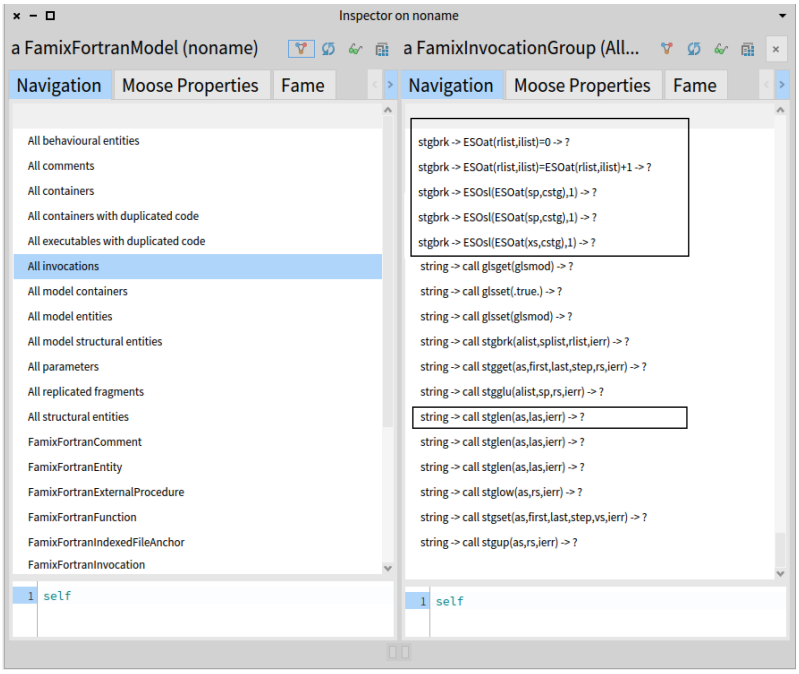


Figure 12 – Les invocations contenues dans le modèle

7 Bilan du stage

G4.1 : Durant les premiers mois de ce stage, j'ai étudié et identifié des analyseurs syntaxiques. J'ai également créé un méta-modèle. Ces missions m'ont permis de consolider mes connaissances sur le fonctionnement d'un compilateur : de la phase d'analyse lexicale à la phase de génération de code machine. Cela nécessite aussi de savoir comment les structures de données qui sont impliquées à chacune de ces étapes fonctionnent : ce que chaque analyseur prend en entrée et produit en sortie.

G4.2 : J'ai aussi pu définir des grammaires en BNF ou en PetitParser2 en m'inspirant de l'approche des « *islands grammar* » ([Kur] et [Moo01]).

G4.3 : L'utilisation de la méthode de développement dirigé par les tests m'ont permis de m'assurer que le code que je produis correspond à ce qui est attendu. Cela aide à l'amélioration de la qualité logiciel.

G7 : Les enjeux de ma mission ont été divers. Un de ces premiers enjeux a été la gestion du calendrier. Pour cela des points réguliers faits avec mon encadrant et des réunions mensuelles sur le suivi du stage ont été programmés. Au cours de ces réunions j'ai été amené à présenter mon travail ce qui m'a permis de le renforcer par l'expertise des retours que j'ai eu. Ces réunions avec Framatome mais aussi les échanges avec les membres de l'équipe m'ont été très bénéfiques sur la façon de mener un travail de recherche notamment par la lecture d'articles scientifiques et de faire un travail de restitution. La rencontre avec Framatome dans leurs bureaux parisiens a été l'occasion de prendre en compte les enjeux de l'entreprise ainsi que l'importance de ce projet pour son futur. Au cours de cette visite, j'ai rencontré les équipes qui m'ont présenté leurs travaux et le contexte dans lequel s'inscrit ce stage dans le projet global.

G3, G6 : Ce stage m'a amené à découvrir les bases de la syntaxe Fortran77 mais aussi les standards plus récents comme le Fortran90 et 2003. J'ai aussi appris le fonctionnement des langages de programmation, la façon de les définir au travers de grammaires et la façon dont ils sont analysés puis reconnus par la machine. Cela implique de comprendre le fonctionnement des différentes étapes de la compilation [Lau20]. Ce stage m'a aussi permis d'apprendre et d'appliquer les connaissances apprises en cours sur la méta-modélisation dans l'environnement Moose.

G11.1 : M'adapter aux diverses contraintes du projet m'a permis d'acquérir des bonnes pratiques dans le génie logiciel. En effet, lors des revues de codes, j'ai pu avoir des retours instructifs qui m'ont aidé à améliorer ma rigueur. Pour compléter ces bonnes pratiques et autres démarches adoptées, l'accent a été mis sur la qualité du code.

G8 : La sensibilité et l'importance des données échangées nous ont contraint à prendre en compte les enjeux de sécurité. En effet l'ensemble des sources fournies est à diffusion limitée. En concertation avec l'ensemble des parties prenantes, un dépôt logiciel privé a été mis en place pour contenir les sources qui m'ont été transmises pour les besoins du stage. Une liste de diffusion contenant toutes les personnes

impliquées dans le projet a également été créée ; cela permet de communiquer les informations avec les parties prenantes.

G9 : Le personnel des ressources humaines a organisé une journée d'information pour présenter le centre et les équipes à l'ensemble des stagiaires le jeudi 24 février. Cette journée a été l'occasion d'un échange en vue d'informer sur le centre et le déroulement d'un stage. Nous avons été sensibilisés sur les enjeux Responsabilité sociétale des entreprises (RSE) au sein du centre ; notamment en ce qui concerne la gestion des déchets et le respect du tri sélectif mis en place. Un accent est mis sur l'accompagnement et le bien-être de l'ensemble des personnels et stagiaires. Cela se traduit par l'octroi de congés aux stagiaires et de journées de Réduction du temps de travail (RTT). De plus un ordinateur portable ainsi qu'un écran ont été mis à ma disposition pour un meilleur confort dans la réalisation de mon stage.

G11.1 : J'ai mis en œuvre une démarche de gestion de projet à travers l'utilisation de l'outil de gestion de version `git`. Par ailleurs au sein de l'équipe, il est demandé à chaque personne de faire un mail en début de chaque semaine récapitulant les tâches réalisées la semaine précédente et celles envisagées pour la semaine courante ; cela permet un suivi efficace et régulier sur l'état d'avancement du projet. Nous avons aussi eu des périodes de programmation en binôme. Cela permet de bénéficier d'un meilleur partage des connaissances et de son expérience.

G11.2 : Être au sein d'une équipe de recherche et en même temps au contact d'un industriel exige d'adapter sa communication pour chacun des interlocuteurs. J'ai su m'adapter aux deux *mondes* dans les différents échanges que j'ai eu.

G11.3 : J'ai su m'intégrer et évoluer dans une équipe de recherche internationale. Nos échanges durant ma période de Projet de fin d'étude (PFE) et passions communes avec certains membres de l'équipe, nous ont amené à organiser un match de football, le sport étant un puissant vecteur pour la cohésion de groupe. L'engouement pour ce premier match a permis de pérenniser cette activité. D'autre part, j'ai eu l'opportunité de participer à la rencontre annuelle autour du langage Pharo [32] qui s'est déroulée sur 3 jours dans les locaux du centre. Cet évènement à été l'occasion de découvrir la communauté, les projets autour de Pharo. Il s'est terminé par un dîner auquel j'ai été convié, avec l'ensemble des personnes de l'équipe. J'ai ainsi pu rencontrer des anciens étudiants GIS et échanger avec eux sur leurs carrières et le monde de la recherche. Ce dîner a été un vrai moment de convivialité et de cohésion.

G11.5 : Après plusieurs semaines à travailler sur l'analyseur proposé pour ce projet je me suis rendu compte de ses limites et ai entrepris d'en trouver un autre. Grâce à des échanges sur des forums de discussions autour de Fortran, j'ai pu définir les critères de choix d'un « bon » candidat et ai ainsi étudié un certain nombre d'analyseurs. J'ai aussi fait le choix d'utiliser une approche par les « *islands grammar* » plutôt que de trouver toute une grammaire capable de reconnaître du fortran77 et de l'ESOPE.

G14 : La découverte du monde de la recherche dans une équipe passionnée et

32. https://days.pharo.org/

passionnante m'a amené à envisager une poursuite en thèse. Il s'agit d'une thèse CIFRE. L'avantage de ce type de thèse CIFRE par rapport à une thèse académique est la possibilité d'évoluer dans deux environnements : entreprise et en laboratoire de recherche. Par ailleurs, le statut qu'elle offre est avantageux par rapport à une thèse académique. Ce projet qui me motive et m'intéresse sera un bon départ pour candidater dans le domaine de la R&D dans les grands groupes en France ou à l'étranger ce qui me permettrait d'améliorer davantage mes compétences linguistiques.

8 Conclusion

Ce stage a été pour moi une expérience très satisfaisante et enrichissante. Faire partie d'une équipe aussi dynamique et accueillante, travailler dans de bonnes conditions sont autant de choses positives que j'en retire.

J'ai pu, grâce à ce stage, améliorer mes compétences et élargir mes connaissances notamment dans le monde de la recherche, en étant confronté à une équipe travaillant sur des projets variés dans le génie logiciel.

Le résultat de ce stage est donc d'avoir identifié deux analyseurs syntaxiques répondant à mes contraintes et besoins comme stipulé dans le cahier des charges. J'ai aussi créé un méta-modèle pour Fortran77, qui a permis de construire le modèle sur lequel je peux effectuer des requêtes.

D'un point de vue personnel, au cours du stage, j'ai beaucoup appris des membres de l'équipe sur les aspects de l'ingénierie logicielle. Cela se matérialise par la participation aux journées des résolutions de bogues organisées le dernier vendredi de chaque mois et les séances de programmation en binôme.

Par ailleurs j'ai pu améliorer mon niveau d'anglais. En effet, compte tenu de l'usage intensif de l'anglais par l'ensemble des membres de l'équipe, j'ai acquis une meilleur compréhension orale qui était mon point faible.

Références

[Fel90] S. I. Feldman. « A Fortran to C converter ». In : *ACM SIGPLAN Fortran Forum* 9.2 (oct. 1990), p. 21-22. ISSN : 1061-7264, 1931-1311. DOI : 10.1145/101363.101366. URL : https://www.netlib.org/f2c/f2c.pdf (visité le 24/02/2022).

[Lev95] G. F. Levy. « Improving the output of the FORTRAN to C translator, f2c ». In : *Software : Practice and Experience* 25.2 (1995), p. 217-227. DOI : https://doi.org/10.1002/spe.4380250207. eprint : https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.4380250207. URL : https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.4380250207.

[Fox+97] Geoffrey Fox et al. « A prototype of Fortran-to-Java converter ». In : *Concurrency : Practice and Experience* 9.11 (1er nov. 1997), p. 1047-1061. ISSN : 1096-9128. DOI : 10.1002/(SICI)1096-9128(199711)9:11<1047::AID-CPE348>3.0.CO;2-V. URL : https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291096-9128%28199711%299%3A11%3C1047%3A%3AAID-CPE348%3E3.0.CO%3B2-V (visité le 22/05/2018).

[Moo01] L. Moonen. « Generating robust parsers using island grammars ». In : *Proceedings Eighth Working Conference on Reverse Engineering*. 2001, p. 13-22. DOI : 10.1109/WCRE.2001.957806.

[Gro+12] Ralf W. Grosse-Kunstleve et al. « Automatic Fortran to C++ conversion with FABLE ». In : *Source Code for Biology and Medicine* 7.1 (mai 2012). DOI : 10.1186/1751-0473-7-5.

[Lau20] Compil Lyon Laure Gonnord. *Cours de Compilation et Analyses de Programmes, ENS de Lyon*. Youtube. 2020. URL : https://www.youtube.com/playlist?list=PLtjm-n_Ts-J-6EU1WfVIWLhl1BUUR-Sqm.

[Kur] Jan Kurs. *Scripting with bounded seas : Extracting Javascript*. URL : https://kursjan.github.io/petitparser2/scripting.html. (accessed : 12.04.2022).

Acronymes

ADT Algebraic data type. 11, 18, 22

ANTLR ANother Tool for Language Recognition. 3, 12–14, 17, 18

AST Arbres de Syntaxe Abstraite - *Abstract Syntax Trees*. 4, 9, 18

AST Arbre de la syntaxe abstraite - *Abstract Syntax Tree*. 1, 2, 6–9, 12, 13, 15, 16, 18, 20, 22

BNF Backus–Naur forme. 14, 24

BNFC BNF Converter. 14, 18

CIFRE Conventions industrielles de formation par la recherche. 26

IDE Integrated Development Environment. 5

Inria Institut National de Recherche en Informatique et en Automatique. 4, 5

OFP Open Fortran Parser. 9, 13, 18

PFE Projet de fin d'étude. 25

R&D Recherche et Développement. 26

RSE Responsabilité sociétale des entreprises. 25

RTT Réduction du temps de travail. 25

UML Unified Modeling Language, ou langage de modélisation unifié. 21

XML Extensible Markup Language. 13

Glossaire

Moose Moose est une plateforme open source pour l'analyse de logiciel et de donnée écrite en Pharo et développée par l'équipe RMOD. 1, 2, 4, 6, 8, 10, 20, 21, 24

Packrat Un analyseur packrat est un type d'analyseur syntaxique qui se base sur la décomposition analytique, et donc découpe un flux continu de caractères puis construit un arbre d'analyse depuis le haut vers le bas. Grâce à cette mémoization, un analyseur packrat peut analyser un grand nombre de grammaires hors-contexte et toutes les grammaires d'expressions (dont celles qui ne représentent pas des langages libres de contexte). [`Wikipedia`]. 16

Pharo Pharo est un langage de programmation pur objet inspiré de Smalltalk. Il est écrit en lui-même et est aussi un environnement de développement. 9, 13, 25