Génération d'ensembles de modèles explorables par couplage de contraintes et de transformation de modèles

Théo Le Calvar[1,2,3] **Fabien Chhel**[2]
Frédéric Jouault[2] **Frédéric Saubion**[1] **Eugene Syriani**[3]
[1] Univ Angers, LERIA, SFR MATHSTIC, 49000 Angers,France
[2] ERIS Team, ESEO Group, Angers, France
[3] DIRO, Université de Montréal, Montréal, Canada
prénom.nom@{univ-angers.fr,eseo.fr}, nom@iro.umontreal.ca

Résumé

Cet article présente les travaux réalisés durant les projets COMOG et Optimodon. Ces travaux s'inscrivent dans les thématiques de l'ingénierie dirigée par les modèles (IDM) et y intègrent des notions de programmation par contraintes. Ces travaux se sont concrétisés par le développement d'ATLc, une extension d'ATL, un langage de transformation de modèles, intégrant des contraintes.

ATLc repose sur la notion d'exploration d'ensembles de modèles. L'exploration d'ensemble de modèles permet à l'utilisateur, grâce à l'utilisation d'ATLc, de générer un ensemble de modèles valides défini en intention par des contraintes.

Dans cet article nous présentons les changements apportés à ATLc au travers de plusieurs cas d'études. Ces changements se concentrent sur deux axes majeurs. Ajouts de sucres syntaxiques au langage pour faciliter l'utilisation. Ajout d'une seconde cible d'exécution, les navigateurs web, par l'utilisation de SVG et de JavaScript. Cette seconde plateforme d'exécution ajoute de nombreuses petites améliorations et est spécialisée dans les diagrammes interactifs.

Abstract

This article presents works realised during COMOG and Optimodon projects. These works integrate constraint solving into model driven ingineering (MDE) and more precisely into model transformation. They materialized with the developement of ATLc, an extension of ATL, a model transformation langage, which integrate constraints.

ATLc uses the model set exploration concept. Model set exploration allows user to generate set of valid models and navigate into the set thank to the use of constraints solver.

In this article we present recent changes to ATLc thourh several case studies. These changes focus on two key points. First, improvements to the langage itself, making it easier to use. Second, adding another execution target to ATLc, the first versions were only running on JavaFX and JVM. This new version allows user to compile their transformations to SVG and JavaScript. This new ATLc compiler is specialized to generate interactive diagrams and feature many quality of life improvements other the older compiler.

1 Introduction

La transformation de modèles s'est imposée comme une technique clef de l'ingénierie dirigée par les modèles (IDM). L'IDM est une branche du génie logiciel prônant l'utilisation de modèles tout au long du cycle de production de systèmes informatiques. L'utilisation de modèles permet d'adapter les concepts manipulés par l'utilisateur à la tâche qu'il doit effectuer, réduisant ainsi la complexité de cette tâche. De plus, l'IDM recommande l'automatisation des traitements des modèles lorsque cela est possible.

La transformation de modèles permet l'automatisation du traitement des modèles et il est ainsi possible d'automatiser le raffinement de modèles abstraits vers de nouveaux modèles plus concrets. Ceci offre la possibilité de travailler sur des modèles de haut niveau d'abstraction qui seront, par l'application d'une suite de transformations, automatiquement raffinés en modèles concrets, pouvant aller jusqu'à la production du code spécifique à une plateforme d'exécution donnée.

Il existe de nombreuses approches de transformation de modèles offrant différents avantages en fonction des utilisations visées (comme l'incrémentalité, la bidirectionalité, ou la *lazyness*). Cependant, la majorité de ces approches se limitent à la génération d'un unique modèle cible pour un modèle source donné. Lorsque plusieurs modèles cibles peuvent correspondre à unique modèle source, il peut être pertinent d'avoir plutôt accès à cet ensemble de modèles.

Avec ATLc et l'exploration de modèles, nous proposons une approche permettant à une transformation de modèles de générer des ensembles de modèles explorables grâce à l'application de contraintes à un modèle partiellement instancié. Ces contraintes permettent de définir en intention l'ensemble des modèles valides et un solveur de contraintes permet de calculer ces solutions. De plus, nous permettons à l'utilisateur de suggérer des changements au modèle qui lui est présenté. Ces changements sont validés par le solveur de contraintes qui pourra, au besoin, réparer le modèle pour assurer la validité des contraintes.

Nous illustrerons différents aspects d'ATLc/web, une nouvelle implémentation d'ATLc permettant l'exécution sur des plateformes web grâce à l'utilisation de SVG et JavaScript, au travers de plusieurs exemples de transformations. Plus de détails sur l'approche et l'implémentation en Java sont disponibles dans [6].

La résolution des contraintes générées est assurée par Cassowary.js [1], une implémentation en JavaScript du solveur linéaire incrémental Cassowary [5]. Cassowary repose principalement sur l'algorithme du simplexe et offre en pratique de très bonnes performances.

Dans les sections 2 et 3 nous illustrons les capacités d'ATLc au travers de deux exemples de diagrammes (respectivement le diagramme de features et le diagramme de séquences) pouvant s'exécuter sur le web. Nous concluons dans la section 4.

Tous les outils présentés dans cet article ainsi que les exemples sont disponibles à [2]. De plus, les exemples présentés dans cet article sont disponibles directement à [3].

2 Diagramme de features

La Figure 1 montre un diagramme de features généré par la transformation. En plus de la version statique visible via la démonstration [3], une version éditable en ligne est disponible [4].

La transformation de modèles avec ATLc reprend les mêmes bases qu'en ATL classique. Une transformation est composée d'un ensemble de règles spécifiant un ou plusieurs éléments à transformer (section `from`) et un ou plusieurs éléments à créer (section `to`). Pour chaque élément créé, il est possible de spécifier les valeurs qui devront être affectées à ses différentes propriétés en utilisant des *bindings* (notés avec l'opérateur `<-`). ATLc reprend la même syntaxe, mais interprète les expressions des contraintes en sorties comme des contraintes et non comme des expressions booléennes.

Avec cette nouvelle version d'ATLc nous permettons l'exécution des transformations dans un navigateur web récent. Pour cela nous compilons les règles ATLc en éléments SVG (`<defs>`) et en code JavaScript servant à créer les contraintes. Ces patrons peuvent ensuite être instanciés par l'ajout d'élément SVG (`<use>`) paramétrables. Grâce à cela il est possible d'intégrer les diagrammes générés avec ATLc dans n'importe quelle page web.

Une des améliorations majeures de cette nouvelle version d'ATLc est la possibilité d'intégrer directement du SVG dans la transformation. Par exemple, dans le Listing 1 il n'y a que deux éléments de sortie (`t` l.5-12 et `cstr` l.13-25). Cependant, le groupe `t` utilise du SVG pour ajouter des éléments de sortie (l. 7-11) sans la lourdeur de l'ATL classique. Il est possible de spécifier des valeurs pour les attributs ou contenus des éléments spécifiés en SVG grâce à l'annotation `${propSource}`.

```
1  unique lazy rule Feature {
2    from
3      s : FeatureDiagram!Feature
4    to
5      t: SVG!Group (
6        class <- 'feature',
7        content <- '
8          <rect id=".box" />
9          <rect id=".outline" />
10         <text id=".label">${s.label}</text>
11       '
12     ),
13     cstr: Constraints!ExpressionGroup (
14       constraints <-
15         outline.width = label.width + thisModule
               .MARGIN
16     and outline.height = label.height +
           thisModule.MARGIN
17     and outline.center = label.center
18     and outline.topLeft.stay('medium')
19     and box.mustContain(outline)
20     and box.center.x = outline.center.x
21     and box.width = 0 -- strong
22     and box.height = 0  -- strong
23     and box.x >= 0
24     and box.y >= 0
25     )
26 }
```

Listing 1: Règle de transformation ATLc pour une `Feature`

La seconde règle de cet exemple, visible dans le Listing 2, illustre quant à elle les améliorations apportées au dialecte de contraintes. Ce nouveau compilateur ATLc impose aux transformations un typage bien plus strict que les versions précédentes. La notation des types en ATL peut être fastidieuse, notamment lors du typage du résultat de l'application d'une règle. Pour

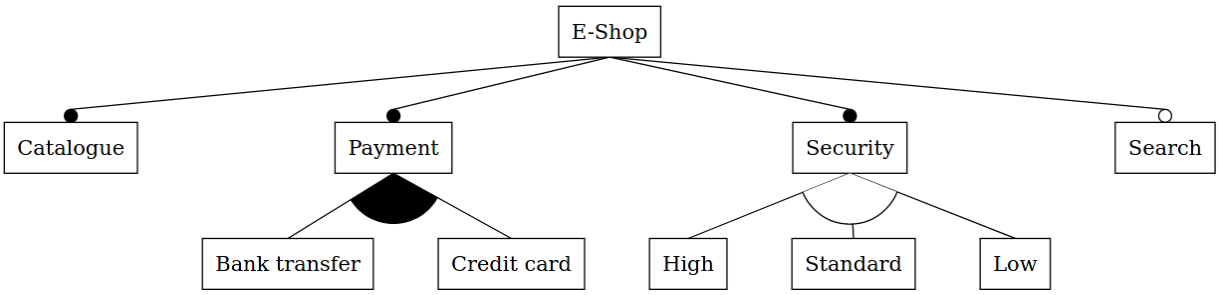

FIGURE 1 – Capture d'écran d'un diagramme de feature généré par ATLc

faciliter ceci, nous avons ajouté un type spécial, marqué `INFER!TYPE` (l.16), indiquant au compilateur d'inférer le type par rapport au contexte.

Dernière particularité de cet exemple. Les lignes 26 à 38 montrent un exemple d'expressions OCL complexes gérées par cette nouvelle version d'ATLc. Cette expression utilise de nombreuses opérations non supportées dans les anciennes versions. On notera l'utilisation des opérations `zipWith` et `forAll` non standard, mais ajoutée ici par soucis pratique.

3 Diagramme de séquences

La Figure 2 montre un exemple de digramme de séquence généré par ATLc. Comme pour le diagramme de *feature*, la transformation complète est accessible [2], ainsi que diagramme de la Figure 2 [3].

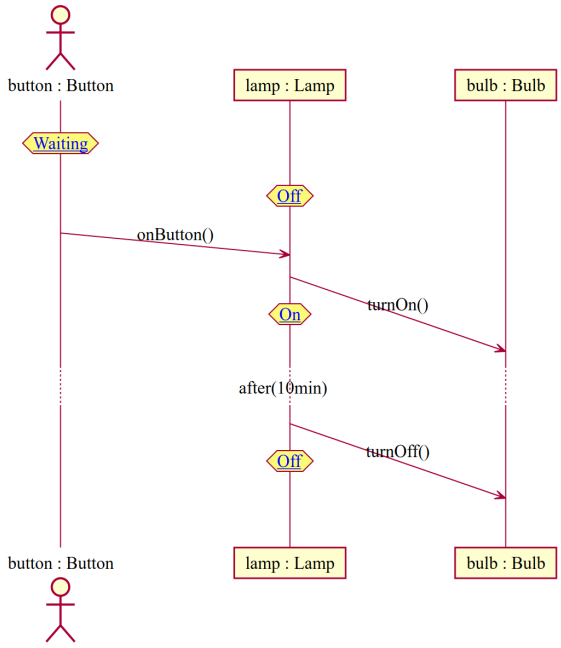

FIGURE 2 – Capture d'écran d'un diagramme de séquence généré avec ATLc

Cette nouvelle version d'ATLc permet l'utilisation de contraintes dans des *helpers* ATL. Un *helper* ATL peut être vu comme une fonction pouvant être appelée dans les règles de transformation. Le Listing 3 montre un *helper* ATL générant des contraintes. Ceci se fait par l'ajout d'une annotation en commentaire avant le *helper* (l.1). L'utilisation de *helpers* permet de factoriser le code de contraintes tout au long de la transformation. Il est possible d'appeler des *helper* depuis un *helper* (l.18).

Ce *helper* illustre aussi le support étendu d'OCL dans les expressions de contraintes. En effet, cette nouvelle version d'ATLc permet l'utilisation de structures conditionnelles dans les contraintes. Nous avons deux structures conditionnelles, le `if` (non utilisé dans ce *helper*) et le `implies` (l.6).

L'ajout de `if` permet de conditionner l'ajout de contraintes par une condition sur le modèle source. Cependant, cette condition n'est pas envoyée au solveur de contraintes, mais est gérée avant, par le code JavaScript postant les contraintes. Ainsi il est possible de contrôler finement l'activation de contraintes sur des solveurs ne le supportant pas.

Le `implies` est un sucre syntaxique pour les `if` sans `else` (ATL impose un `else` aux `if`).

4 Conclusion

Dans cet article nous avons présenté une version d'ATLc pouvant générer des diagrammes interactifs sur le web. Ces diagrammes utilisent SVG et JavaScript et sont donc facilement intégrables dans une page web. La résolution des contraintes est assurée par Cassowary. Nous travaillons actuellement à l'intégration de diagrammes spécifiés avec ATLc dans l'éditeur Gentleman [7] comme primitives de projection et ainsi permettre une plus grand liberté que ce qui serait uniquement faisable en HTML.

La nature déclarative d'ATLc nous a permis de facilement développer ce second environnement d'exécution adapté au web. De plus, nous avons profité de

```
unique lazy rule Alternative {
  from
    s: FeatureDiagram!Alternative
  to
    t: SVG!Group (
      class <- 'arc',
      content <- '
        <clipPath id=".clipPath">
          <polygon id=".poly" />
        </clipPath>
        <circle id=".outline" fill="${s.fill}"
            stroke="${s.stroke}" clip-path="url
            (#%this%.clipPath)"/>
      '
    ),
    cstr: Constraints!ExpressionGroup (
      constraints <-
        let parent : INFER!TYPE =
          thisModule.Feature(s.arcs->first().
              parent) in
        let firstChild : INFER!TYPE =
          thisModule.Feature(s.arcs->first().
              child) in
        let lastChild : INFER!TYPE =
          thisModule.Feature(s.arcs->last().
              child) in

          outline.center = parent.outline.
              bottom.center
        and ...
        and (
          let childArcs : INFER!TYPE = s.arcs->
              collect(arc|
            thisModule.Feature(arc.child)
          )
          in
          Tuple {
            left = childArcs,
            right = childArcs->subSequence(2,
                childArcs->size())
          }->zipWith(cur, next |
              cur.box.topRight.x
            + thisModule.HORIZONTAL_SEPARATION
            =
              next.box.x
          )->forAll(e | e)
        )
    )
}
```

Listing 2: Extrait d'une règle de transformation ATLc pour une `Alternative`

la flexibilité de cette nouvelle plateforme d'exécution pour ajouter de nouvelles fonctionnalités à ATLc. Ces nouveautés, comme la possibilité d'intégrer du SVG ou un meilleur support d'OCL dans la spécification des contraintes, facilitent l'utilisation du langage en réduisant la complexité globale de la transformation.

Tous ces outils, ainsi que les exemples de cet article, sont libres et disponibles [2].

Remerciements

Ces travaux sont financés par le projet de la région des Pays de la Loire RFI Atlanstic 2020.

```
-- @constraints (
helper def: participantConstraints(part : Seq!
    Participant) : Boolean =
  let tgt : INFER!TYPE = thisModule.Participant(
      part) in
  let afterTgt : INFER!TYPE = thisModule.
      Participant(part.after)
  in
  ((not part.x.oclIsUndefined()) implies (
      tgt.outline.x = part.x
    and tgt.outline.y = part.y
  ))
  and ((not part.after.oclIsUndefined()) implies
      (
    tgt.outline.bottom.p1.y = afterTgt.outline.
        bottom.p1.y
    and tgt.outline.center.x >= afterTgt.outline
        .center.x
      + (tgt.outline.width + afterTgt.outline.
          width) / 2
      + thisModule.INTER_OBJECT_MARGIN
    and tgt.line.p2.y = afterTgt.line.p2.y
  ))
  and ...
  and thisModule.textOutline(tgt.outline, tgt.
      name)
  and ...;
```

Listing 3: Extrait d'un *helper* contenant des contraintes

Références

[1] Cassowary.js project page. https://github.com/slightlyoff/cassowary.js. 2021-05-14.

[2] Code source des compilateur atlc et autres outils liés à atl. https://github.com/ESEO-Tech/ATL-Tools-Library. 2021-06-04.

[3] Démo en ligne de diagrammes générés avec atlc. https://eseo-tech.github.io/ATL-Tools-Library/atlc/. 2021-06-04.

[4] Éditeur de feature diagram en ligne utilisant une transformation atlc. https://aof.page.kher.nl/feature-diagram/. 2021-06-04.

[5] Greg J Badros, Alan Borning et Peter J Stuckey : The Cassowary linear arithmetic constraint solving algorithm. *TOCHI*, 8(4):267–306, 2001.

[6] Théo Le Calvar, Fabien Chhel, Frédéric Jouault et Frédéric Saubion : Coupling solvers with model transformations to generate explorable model sets. *Software and Systems Modeling*, feb 2021.

[7] Louis-Edouard Lafontant et Eugene Syriani : Gentleman. *In Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems : Companion Proceedings*. ACM, oct 2020.