Un Framework dirigé par les modèles pour l'extraction de règles métier à partir d'applications COBOL

Valerio Cosentino, Philippe Bauquel, Jacques Perronnet, Patrick Albert, Jordi Cabot

Un Framework dirigé par les modèles pour l'extraction de règles métier à partir d'applications COBOL

Valerio Cosentino[1,2], Philippe Bauquel[3], Jacques Perronnet[3], Patrick Albert[2] and Jordi Cabot[1]

[1] AtlanMod, INRIA & EMN, Nantes, France
[2] IBM CAS France
[3] IBM, France
{albertpa, bauquel.p, jacques_perronnet, valerio.cosentino}@fr.ibm.com,
jordi.cabot@mines-nantes.fr

résumé - abstract

Les entreprises s'appuient sur la logique métier encodée dans leurs systèmes d'information. Ce sont souvent des applications existantes qui ne sont pas conçues pour représenter et opérationnaliser cette logique indépendamment des aspects techniques. Par conséquent, ces systèmes ne sont pas capables de répondre rapidement aux nouvelles exigences requises par le marché. La découverte et la gestion de la logique métier dans les applications sont des activités qui consomment beaucoup de temps et qui sont sources d'erreurs, car à chaque fois le code doit être analysé, modifié et testé. Afin de simplifier ces activités, ce papier propose un framework dirigé par les modèles pour l'extraction de la logique métier. L'Ingénierie Dirigée par les Modèles est utilisée pour découvrir et représenter la logique extraite en fournissant des artefacts en sortie qui aident à sa compréhension.

mots clés: Ingénierie Dirigée par les Modèles, Règles Métiers, Extraction

Companies rely on the logic embedded in their Information Systems; that are often legacy applications not designed to represent and operationalize business logic independently from the technical aspects of the applications. Consequently, legacy systems are not able to respond quickly to the new requirements constantly demanded by the market. Discovering and tuning business logic in a legacy system, are time-consuming and error-prone activities, since each time the source code must be analysed, modified and tested. In order to simplify these activities, this paper proposes a model driven engineering framework for extracting business logic. Model Driven Engineering is used to ease the discovery and representation of the extracted logic providing understandable output artifacts.

keywords: Model Driven Engineering, Business Rules, Extraction

1 Introduction

La plupart des entreprises dépendent d'applications existantes, implémentées au long des quarante dernières années. Nous nous concentrons sur celles écrites en COBOL (COmmon Business-Oriented Language), qui est un des languages de programmation les plus anciens et qui continue à jouer un rôle vital pour les entreprises avec ses 220 milliards de lignes de code source encore actives. Ces systèmes contiennent de la logique métier critique, qui doit être préservée et entretenue au fil du temps; mais qui est, en même temps, difficile à analyser et à modifier, car elle est dispersée dans le code source. Cette complexité rend les applications difficilement adaptables pour répondre aux besoins du marché. D'autre part, les entreprises ne veulent pas migrer ces anciens systèmes vers des technologies plus récentes en raison du coût souvent prohibitif des processus de ré-ingénierie. Par conséquent, nous proposons une méthode automatique et

non-intrusive pour l'extraction de règles métier à partir de systèmes existants sans forcer leur complète réécriture.

Le processus d'extraction des règles métier à partir d'un système d'information est appelé BREX (Business Rule EXtration)[10]. Il est principalement composé de trois étapes: l'identification des variables du métier, l'identification des règles métier et la représentation de ces dernières. La première étape est utilisée pour localiser, dans le code source, les variables qui pointent sur des concepts du métier. La deuxième étape identifie les règles de gestion à l'aide de techniques de *program slicing* [11] en fonction des variables découvertes dans l'étape précédente. La troisième étape présente les règles extraites à l'aide d'artefacts compréhensibles (textes, graphes, ...). Une caratéristique importante dans un processus BREX est la traçabilité entre les règles extraites et le code source correspondant. C'est un élément clé qui permet d'expliquer et justifier l'origine des règles métier extraites.

Une règle métier, selon le *Business Rule Group*[1], est une déclaration qui contraint un aspect du métier indépendamment des aspects techniques. Au niveau du code source, une règle métier peut être considérée en fonction des données[6] qu'elle utilise ou aux opérations conditionnelles qui la composent[13].

Cet article décrit un framework modulaire dirigé par les modèles pour l'extraction des règles de gestion à partir d'applications COBOL. Nous montrons que l'Ingénierie Dirigée par les Modèles[8] (IDM), appliquée à la rétro-ingénierie et aux méthodes BREX, offre des avantages importants par rapport aux autres approches. L'IDM nous permet de construire un framework composé en plusieurs étapes modulaires, puisque chacune d'elles est liée aux autres par ses modèles en entrée et sortie. L'IDM permet de travailler avec une représentation plus abstraite et homogène du système, ce qui évite les problèmes liés aux spécificités techniques et fournit une solution non-intrusive. De plus, lors de la représentation du système sous forme de modèles, nous pouvons profiter du grand nombre d'outils de l'IDM pour manipuler le système. Cet article est structuré comme suit: la Section 2 présente la description du framework, la Section 3 décrit un cas d'étude, la Section 4 analyse les travaux associés et la Section 5 termine le papier avec la conclusion et les travaux futurs.

2 Description du Framework

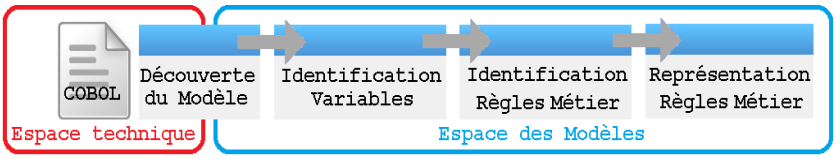

Figure 1: vue d'ensemble du Framework

Le framework (fig. 1) est composé par les trois principales étapes d'un processus BREX (Identification des Variables, Identification et Représentation des Règles Métier) plus une étape supplémentaire utilisée pour passer de l'espace technique défini par le language du système à l'espace des modèles. Cette dernière étape est realisée en utilisant le COBOL Application Model[2] d'IBM Rational Developer for System z[3] (RDZ).

[1] http://www.businessrulesgroup.org/first_paper/br01c1.htm

[2] http://pic.dhe.ibm.com/infocenter/ratdevz/v8r5/index.jsp?topic=%2Fcom.ibm.rsar.analysis.codereview.cobol.doc%2Ftopics%2Fcac_customrule_camapi.html

[3] http://publib.boulder.ibm.com/infocenter/ieduasst/rtnv1r0/index.jsp?topic=/com.ibm.iea.rdz/

Le framework peut fonctionner avec les outils fournis par IBM Rational Programming Patterns for System z[4] (RPP). RPP est fortement basé sur les principes de l'IDM, qui facilitent l'intégration avec le framework. RPP permet aux experts du domaine de définir un système à travers des modèles qui représentent les données, leurs relations et leurs descriptions.

L'identification des Règles Métiers fournit les moyens pour identifier les règles de gestion à partir d'une variable. Les entrées de cette étape sont le modèle d'un programme COBOL et le nom d'une variable dans celui-ci. Elle est composée par deux sous-étapes: l'Analyse du Flux de Contrôle du programme et l'opération de Découverte de Règles. La sortie de cette étape est un graphe de flot de contrôle[1] (GFC) enrichi avec les informations concernant les règles trouvées.

La Représentation de Règles Métiers produit des artefacts qui décrivent la logique Métier extraite. Elle est composée par deux sous-étapes: l'Extraction du Vocabulaire et la Visualisation des Règles. Elle prend en entrée le résultat de l'étape précédente et le modèle de l'application. Elle génère des artefacts textuels et des graphes. Les premiers sont utilisés pour l'analyse de chaque règle, tandis que les seconds se concentrent sur leur orchestration.

Les étapes d'Identification des Variables, d'Identification et de Représentation des Règles Métier sont décrites dans les sections suivantes. Elles ont été implémentées par une chaîne de transformations de modèles en ATL Transformation Language (ATL)[4].

2.1 Identification des Variables

L'étape d'Identification des Variables est utilisée pour réduire le nombre de variables à analyser en filtrant celles qui ne sont pas liées au métier. Dans ce framework, cette phase peut être soit manuelle soit automatique. Dans le premier cas, l'utilisateur accède au code et choisit une variable. Il peut utiliser les outils que RPP offre pour naviguer parmi les modèles des données, en identifiant ceux qui correspondent à des concepts métier.

Dans le second cas, nous avons défini un ensemble d'heuristiques afin d'identifier les variables qui sont modifiées dans les déclarations qui contiennent des opérations mathématiques. Pour ce faire, les déclarations ont été classées en plusieurs catégories: *IfThenElse*, *IfThen*, *IfThenElseGoTo*, *IfNextElseGoTo*, *IfThenGoTo*, *GoTo*, *Move*, *Perform*, *Call*, *Computation* qui contient toutes les commandes mathématiques (ADD, SUBTRACT, ...) et *End* qui inclut les commandes pour terminer un programme (STOP RUN, GO BACK, ...).

Les variables contenues dans chaque déclaration sont rassemblées en quatre catégories. *Les variables de condition* sont les variables qui apparaissent dans les conditions des déclarations conditionelles. *Les variables d'index* sont les indices des tableaux. *Les variables source* et *les variables cible* représentent respectivement les variables qui participent et sont modifiées dans une déclaration. A la fin de cette étape, les noms des *variables cibles* dans les *statement* classés comme *Computation* sont transmis automatiquement, un par un, à l'étape d'Identification de Règles Métier.

2.2 Identification de Règles Métier

L'Identification de Règles Métier est composée de deux sous-étapes: l'Analyse du Flux de Contrôle du programme et l'opération de Découverte de Règles. Elle prend en entrée le modèle d'un programme COBOL et produit en sortie un GFC pour ce programme en ajoutant des informations supplémentaires pour les déclarations qui composent les règles de gestion identifiées dans le modèle d'éntrée. l'Analyse du Flux de Contrôle crée un GFC d'un programme COBOL

[4] `http://publib.boulder.ibm.com/infocenter/rppzhelp/v8r0/index.jsp`

donné; tandis que l'opération de Découverte de Règles trouve les déclarations qui composent les règles de gestion par rapport aux variables identifiées dans l'étape précédente.

Analyse du Flux de Contrôle. Cette étape génère un GFC d'un programme COBOL donné. En plus, pour chaque déclaration, les informations concernant le type de la déclaration et les variables qu'elle contient sont importées de l'étape d'Identification des Variables.
L' Analyse du Flux de Contrôle prend en entrée le modèle d'un programme COBOL et retourne un modèle représentant le GFC, qui sauvegarde également les liens entre les deux modèles. Cette dernière opération implémente la traçabilité dans le cadre de l'IDM[5], qui est présente dans toutes les étapes du framework. Cette traçabilité est utilisée pour lier les éléments du code source à ceux qui composent les règles de gestion extraites (traçabilité BREX[2]).

Découverte de Règles Métiers. Cette étape recherche les déclarations qui composent les règles métier par rapport à une variable dans le programme. Selon cette variable, les déclarations sont rassamblées, regroupées et ordonnées via une technique de *program slicing*. Les entrées de cette étape sont le GFC d'un programme et une variable. La sortie est le GFC enrichi avec les informations concernant les règles de gestion liées à la variable en entrée.

La technique de *slicing* est constituée de trois phases. Tout d'abord, les déclarations, qui contiennent la variable passée en entrée, sont extraites. Dans cet ensemble, les *if-statements* peuvent être sélectionnés dans le cas où la variable apparaît dans des branchements *then* ou *else*, et non dans la condition. Dans la seconde phase, en fonction des informations contenues dans le GFC, une matrice de connectivité entre toutes ces déclarations est calculée. Deux déclarations sont reliées dans la matrice si un chemin dans le programme existe entre elles. Le chemin ne doit pas contenir d'autres déclarations qui font partie de la matrice. Le coût de l'initialisation de la matrice est égal à celui de la Recherche en Largeur. La matrice de connectivité est utilisée pour identifier toutes les règles métier correspondant à une variable donnée. Chaque règle est composée d'un sous-ensemble de déclarations. Dans la dernière phase, les conditions qui déclenchent les déclarations de chaque sous-ensemble sont récupérées et ajoutées à la règle métier correspondante.

2.3 Représentation des Règles Métier

La Représentation des Règles Métier fournit des artefacts compréhensibles qui décrivent la logique métier extraite et son orchestration. Cette étape est composée de deux opérations: l'Extraction du Vocabulaire et la Visualisation des règles. Ses entrées sont le modèle du programme COBOL, son GFC enrichi avec les informations ajoutées lors de l'étape précédente et, éventuellement, le vocabulaire de l'application défini par l'utilisateur. La sortie est composée des textes et graphes qui facilitent la compréhension de la logique extraite.

L'Extraction du Vocabulaire. Cette étape définit des descriptions pour les structures du language et de données de l'application. Il s'agit d'une opération manuelle, dans laquelle la qualité du vocabulaire en sortie dépend de la clarté des descriptions écrites par l'utilisateur. Les applications RPP permettent à l'utilisateur de joindre à tous les modèles des données une étiquette contenant une brève explication. RPP fournit une interface[5] qui rend possible la navigation parmi ces modèles qui composent le système. De cette manière, il est possible de récupérer automatiquement les étiquettes et de les associer aux entités correspondantes. Ce processus permet d'extraire le vocabulaire pour des applications RPP.

La Visualisation des Règles. Cette étape fournit des artefacts qui facilitent la compréhension des règles extraites et les relations entre elles. Ses entrées sont le modèle

[5] `http://publib.boulder.ibm.com/infocenter/rppzhelp/v8r0/index.jsp?topic=%2Fcom.ibm.pdp.pac.doc%2Fjavadocs%2FJavaDocMaf%2Findex.html`

du programme COBOL, le GFC correspondant avec les informations de la logique métier, et éventuellement, le vocabulaire de l'application. Les sorties de cette étape sont représentées par du texte et des graphes utiles aux utilisateurs techniques ou à ceux du métier. Les sorties textuelles offrent une représentation de chacune des règles en utilisant le code source et éventuellement le vocabulaire, tandis que les graphes montrent l'orchestration des règles.

3 Cas d'Étude

Afin d'illustrer notre framework, nous fournissons un petit programme COBOL contenant plusieurs règles de gestion. Dans le cas d'étude[6], un client peut acheter des produits dans un magasin si celui-ci est ouvert et en fonction de ses besoins, son argent et de la capacité de son sac. Le magasin propose plusieurs produits alimentaires, chacun d'entre eux est représenté par un prix unitaire et la quantité disponible correspondante. En fig.2 l'orchestration entre les règles concernant les variables *money* (à gauche), *pr-veg* et *pr-fruit* (à droite) est montrée.

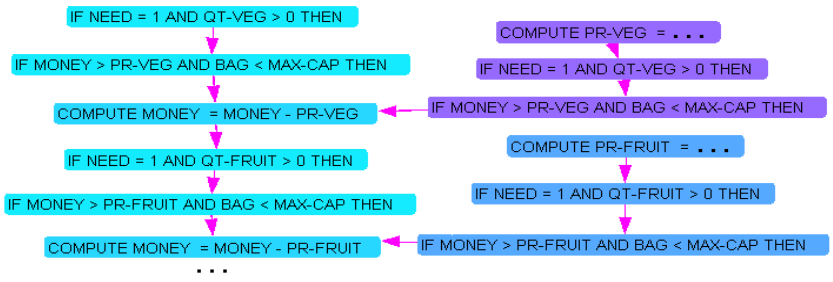

Figure 2: Exemple d'orchestration entre règles métier

4 Travaux Associés

La découverte de règles métier à partir du code source est un domaine de recherche qui a été abondamment étudié. La plupart des travaux précédents est basée sur les opérations de *slicing*, comme en [10], qui décrit une méthode pour l'analyse du code et l'identification de la logique métier, sans fournir les étapes d'identification des variables et de présentation des règles métiers.

Les techniques de *slicing* sont généralement utilisées par les frameworks BREX. [9], [14], [7], par exemple, présentent des frameworks contenant les trois étapes que nous avons décrites dans ce papier. Dans ces travaux, la traçabilité entre le code source et les règles de gestion et l'orchestration des règles découvertes ne sont pas traitées. Grâce à l'IDM, nous pouvons offrir un mécanisme de traçabilité complet. En outre, notre framework produit des artefacts qui permettent de montrer l'orchestration des règles extraites. Notre framework, à l'aide des techniques de l'IDM, fournit de la modularité, de l'extensibilité et une séparation entre la représentation interne et externe des règles métier.

Dans le cadre de l'IDM, de nombreux travaux de rétro-Ingénierie existent. Parmi eux, [12], [3] se concentrent sur l'extraction de connaissances dans des modèles à partir de l'information non-structurée des systèmes existants afin de faciliter la migration vers de nouvelles technologies. Dans ces deux travaux, les règles métier font partie de la connaissance découverte, mais la manière dont elles sont identifiées n'est pas explicitement décrite.

[6] `http://docatlanmod.emn.fr/CIEL2013`

5 Conclusion & Travaux Futurs

Dans cet article, nous avons présenté un framework dirigé par les modèles pour l'extraction des règles métier à partir d'applications COBOL. Les informations relatives aux règles de gestion sont sauvegardées dans un GFC, qui est ensuite utilisé comme entrée pour l'étape de présentation. Les artefacts de sortie du framework sont utilisés pour comprendre les règles et l'orchestration entre elles. En outre, toutes les règles sont liées à la partie correspondante du code source par un mécanisme de traçabilité. La modularité et le haut niveau d'abstraction fourni par l'IDM permettent au framework de fonctionner avec des applications COBOL génériques ou d'un fournisseur donné. Ceci a été démontré en intégrant le framework avec des applications RPP. Le framework est automatique, mais permet aussi à l'utilisateur de fournir manuellement des variables à analyser. En outre, l'utilisateur peut définir un vocabulaire pour les données qui composent les règles de gestion extraites.

Le framework a été testé avec succès sur une application RPP contenant 14 programmes et 130 variables, soit plus de 6500 lignes de code. Grâce aux outils offerts par RPP, 30 variables ont été identifiées et les règles de gestion découvertes ont été soumises aux experts de l'application. Ils ont validé les résultats et leurs retours ont été positifs par rapport à l'utilité de l'information récupérée. À l'avenir, nous envisageons de tester notre framework avec des systèmes plus complexes et de l'étendre en analysant la couche de persistance des applications, en particulier les contraintes codées dans les *triggers* SQL.

References

[1] Frances E. Allen. Control flow analysis. *SIGPLAN Not.*, 5(7):1–19, 1970.

[2] Hendryx S. Baxter I. A standards-based approach to extracting business rules. `http://www.semdesigns.com/Company/Publications/ExtractingBusinessRules.pdf`.

[3] Barbier F., Deltombe G., Parisy O., and Youbi K. Model driven reverse engineering: Increasing legacy technology independence. In *IWRE*, pages –, 2011.

[4] Jouault F., Allilaire F., Bézivin J., and Kurtev I. Atl: A model transformation tool. *Sci. Comput. Program.*, 72(1-2):31–39, 2008.

[5] A. Galvão I., Goknil. Survey of traceability approaches in model-driven engineering. In *EDOC*, pages 313–, 2007.

[6] Aiken P. H., Muntz A. H., and Richards R. Dod legacy systems: Reverse engineering data requirements. *Commun. ACM*, 37(5):26–41, 1994.

[7] Hai Huang. Business rule extraction from legacy code. In *COMPSAC*, pages 162–, 1996.

[8] Bézivin J. Model driven engineering: An emerging technical space. In *GTTSE*, pages 36–64, 2005.

[9] Sneed H. M. Extracting business logic from existing cobol programs as a basis for redevelopment. In *IWPC*, pages 167–175, 2001.

[10] Sneed H. M. and Erdös K. Extracting business rules from source code. In *WPC*, pages 240–, 1996.

[11] Weiser M. Program slicing. *IEEE Trans. Software Eng.*, 10(4):352–357, 1984.

[12] Rodríguez-Echeverría R., Conejero J. M., Clemente P. J., Preciado J. C., and Sánchez-Figueroa F. Modernization of legacy web applications into rich internet applications. In *ICWE Workshops*, pages 236–250, 2011.

[13] Jarzabek S. Life-cycle approach to strategic re-engineering of software. *Journal of Software Maintenance*, 6(6):287, 1994.

[14] Wang X., Sun J., Yang X., He Z., and Maddineni S. Business rules extraction from large legacy systems. *Software Maintenance and Reengineering*, 0:249, 2004.