Extracting Business Rules from COBOL: A Model-Based Tool

Valerio Cosentino
AtlanMod, INRIA, EMN, Nantes, France
and IBM France
valerio.cosentino@fr.ibm.com

Jordi Cabot
AtlanMod, INRIA, EMN, Nantes, France
jordi.cabot@mines-nantes.fr

Patrick Albert
Philippe Bauquel
Jacques Perronnet
IBM France
albertpa, bauquel.p
jacques_perronnet@fr.ibm.com

***Abstract*—This paper presents a Business Rule Extraction tool for COBOL systems. Starting from a COBOL program, we derive a model-based representation of the source code and we provide a set of model transformations to identify and visualize the embedded business rules. In particular, the tool facilitates the definition of an application vocabulary and the identification of relevant business variables. In addition, such variables are used as starting point to slice the code in order to identify business rules, that are finally represented by means of textual and graphical artifacts. The tool has been developed as an Eclipse plug-in in collaboration with IBM France.**

I. Introduction

Organizations rely on the logic embedded in their Information Systems for their daily operations. This logic implements the business rules[1] in place in the organization, which must be continuously adapted in response to market changes.

Unfortunately, this evolution implies understanding and evolving also the underlying software components enforcing those rules. This is challenging because of two main reasons. Firstly, the business logic is usually scattered throughout the code and intertwined with technical and auxiliary code. Secondly, documentation (textual descriptions, models of the software,...) is generally not available or outdated. This is specially true in the case of legacy systems where the original developers may have left the organization. One particular important type of legacy systems are COBOL systems which still play a critical role in the business world. They are the focus of this work.

In order to facilitate the comprehension and the evolution of COBOL-based systems, we present a Business Rule Extraction (BREX) tool[1], that aims at extracting the (business) logic hard-coded in a system as a set of business rules. These business rules can then be validated (or updated/reimplemented) by the company's stakeholders.

The BREX tool is based Model Driven Engineering (MDE) technologies, that offer a higher abstraction level and an homogeneous (model-based) representation of the system. In addition, it benefits from the plethora of available MDE tools for model manipulation, visualization and transformation (Xtext[2], Portolan[3], ATL[4]).

[1]A demonstration of the tool is available at http://docatlanmod.emn.fr/BrexCobolExample/intro.html

The tool has been developed at IBM France in response to (and in cooperation with) the needs of IBM Rational Software Group to improve the reverse engineering services and tools they offer to their customers. It is integrated within IBM Rational Programming Patterns[2] (RPP).

This paper is structured as follows: Section 2 describes the BREX process embedded in the tool, Section 3 presents the implementation details and Section 4 concludes the paper.

II. BREX Process

The BREX process performed by the tool is presented in [5]. It consists of 4 phases: an initial *Model Discovery* phase plus the three main steps of a standard BREX process[6], *Variable Identification*, *Business Rule Identification*, and *Business Rule Representation*.

The *Model Discovery* step is needed to go from the "programming" (or grammar-ware) technical space to the model-ware space. Given a COBOL program, it creates a model-based representation of the source COBOL code. This initial model has a one-to-one correspondence with the code so there is no information loss at this point. This model will be then manipulated in the next steps to extract the business rules. In particular, *Variable Identification* identifies in the code the variables representing business concepts. *Business Rule Identification* locates business rules using code slicing techniques[7] on the variables found in the previous step. Finally, the *Business Rule Representation* step visualizes the extracted rules.

III. Implementation Details

Three Eclipse wizards allow the user to interact with the tool. In the following, we describe the process to create a new BREX project. It is composed by 5 pages.

Create BREX Project. In the first page, the user defines a name for the new project and indicates the location of the COBOL source code and optionally the COBOL model path. According to the information entered, the tool creates the corresponding BREX project in the workspace and generates a copy of both COBOL program and model in its folder *PROGRAM*. In case the COBOL model path has not been

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

inserted, the model is generated using the model discovery provided by the COBOL Application Model of IBM Rational Developer[3] (RDZ). In addition, the tool creates in the folder *BRI* (i.e., Business Rule Identification) a model representing the Control Flow Graph (CFG) of the input program.

Define/Import Application Vocabulary. In the second page, the user can either define or import an application vocabulary. The former consists of providing a description for any of the variables in the program, while the latter allows to import a vocabulary from an IBM RPP application. In particular, since RPP allows to attach to any data structure (programs, variables, ...) a label containing a short explanation, we provide a way to collect automatically those labels as part of the reverse engineering process and associate them to the corresponding entity. If provided, such vocabulary improves the visualization of the extracted rules. The vocabulary is copied to the folder *BRI*.

Rule Discovery. In the third page, the user can select one or more variables to analyse and add them to the BREX process. The variables can be selected either manually or in an assisted way. In particular, the tool classifies the variables in the program according to their types (e.g., condition, source, target, etc.) and the statements where they appear (e.g., conditional, computational, etc.). Such discovered variables are stored in the folder *VI* (i.e., Variable Identification) of the BREX project.

Finally, the tool runs a *Rule discovery* operation for the selected variables. Such operation relies on program slicing techniques to recover the business rules associated to one or more variables. A rule represents a possible execution path in the program related to the selected variables and it includes one or more statements modifying/accessing such variable(s).

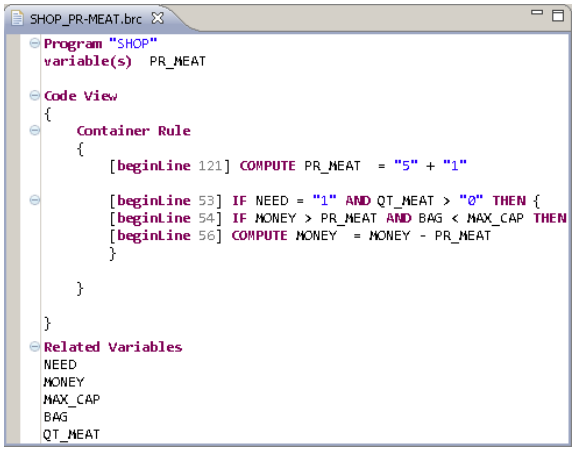

Fig. 1. DSL instance for code-based representation

The output of the *Rule Discovery* is added to the CFG (i.e., folder *BRI*) and exported to the *BRR* (i.e., Business Rule Representation) folder as textual artifacts. These artifacts provide code-based and vocabulary-based representations of the extracted rules. They conform to a couple of Domain Specific Languages (DSL) (e.g., Fig.1) defined using Xtext.

[3]http://publib.boulder.ibm.com/infocenter/ieduasst/rtnv1r0/index.jsp

Context Discovery. In the fourth page, the user can optionally ask the tool to discover the remaining conditions (i.e., Context Discovery) that trigger the rule/s discovered so far. In addition, the user can either select or import code interval(s) where to look for the remaining conditions. In the first case, the user enters manually the line numbers that correspond to the intervals. In the second case, the user can import the information from a RPP application. The output of the *Context Discovery* phase is added to the CFG in the *BRI* folder and exported in the *BRR* folder as textual artifacts.

Graph Generation. In the fifth page, the user can generate graph-based artifacts in order to visualize the relationship between the identified rules in the program. These graph representations (e.g., Fig.2) are added to the *BRR* folder and visualized with Portolan[3], a model-based cartography tool.

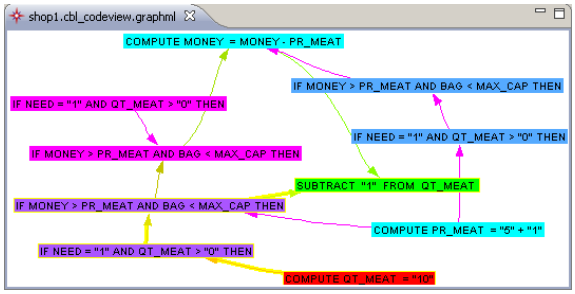

Fig. 2. Graph representation

IV. Conclusion

In this paper, we have presented a tool implementation for extracting and visualizing business rules embedded in COBOL applications. The code is analyzed and sliced to extract the relevant statements for one or more business concepts. Visualization techniques provide a comprehensive output that facilitates the understanding of the business rules enforced in the COBOL application. The tool has been developed within IBM France, that has provided expertise, tools support and a use case.

References

[1] D. Hay, K. A. Healy, and J. Hall, "Defining Business Rules – What Are They Really?" 2000, http://www.businessrulesgroup.org/first_paper/br01c1.htm.

[2] M. Eysholdt and H. Behrens, "Xtext: implement your language faster than the quick and dirty way," in *SPLASH*, 2010, pp. 307–309.

[3] V. Mahe, S. Martinez Perez, G. Doux, H. Brunelière, and J. Cabot, "PORTOLAN: a Model-Driven Cartography Framework," INRIA, Tech. Rep. RR-7542, 2011.

[4] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev, "ATL: A model transformation tool," *Sci. Comput. Program.*, vol. 72, no. 1-2, pp. 31–39, 2008.

[5] V. Cosentino, J. Cabot, P. Albert, P. Bauquel, and J. Perronnet, "Extracting Business Rules from COBOL: A Model-Based Framework," in *WCRE*, 2013, to appear.

[6] H. M. Sneed and K. Erdös, "Extracting Business Rules from Source Code," in *WPC*, 1996, pp. 240–247.

[7] M. Weiser, "Program slicing," *Software Engineering, IEEE Transactions on*, no. 4, pp. 352–357, 1984.