МАТЕМАТИКА И МАТЕМАТИЧЕСКО ОБРАЗОВАНИЕ, 2012
MATHEMATICS AND EDUCATION IN MATHEMATICS, 2012
Proceedings of the Forty First Spring Conference
of the Union of Bulgarian Mathematicians
Borovetz, April 9–12, 2012

EXTRACTING BUSINESS RULES THROUGH STATIC ANALYSIS OF THE SOURCE CODE*

Krassimir Manev, Neli Maneva, Haralambi Haralambiev

The business rules (BR) approach has been introduced at the end of the past century in order to facilitate the specification of business software and to make it more adequate to the needs of the corresponding business. Nowadays most of the stated goals of the approach have been achieved. But the efforts, both scientific and practical, for providing "a rigorous basis for reverse engineering BR from existing systems" are still in progress. This paper describes an approach for deriving BR from source code, based on the methods of source code static analysis. Some advantages and disadvantages of such simplified approach are outlined.

1. Introduction. The Business Rules Project (BRP or GUIDE BRP) started in 1993 with an aim "to formalize an approach for identifying and articulating the rules which define the structure and control the operation of an enterprize". In 2000 the research group issued its first Final report [1]. Underlying in it the fact that each business has a specific set of rules such that partners in the business – businessmen and clients – obey, members of the research team pointed four main objectives of the project, namely:

- to define how to apply BR to information systems;
- to define and describe BR and associated concepts;
- to provide a rigorous basis for engineering new systems based on formal definitions of business rules;
- to provide a rigorous basis for reverse engineering of business rules from existing systems.

Soon after the appearance of the report the first three objectives of the project have been, more or less, achieved. Specifying software system through the preliminary outlined BR is a wide-spread practice nowadays [2]. This paper is dedicated to the last of the above mentioned objectives.

As stressed in the Final report [1] the authors of the BR concept supposed that the approach must be used for reverse engineering of the existing systems. It is well known that software systems get old. There are two possible ways to solve the problems with a legacy application. One of them is to build a new system from scratch. Another – to change the existing system or part of it with respect to some new demands and

*ACM Classification: D.2.7, D.2.5.
Key words: Business Rules, Extraction of BR from Source Code, Static Analysis, Automated Test Data Generation.
This work is supported by the National Scientific Research Fund under the Contract DTK 02-69/2009.

requirements. In both cases it will be very helpful for the development team to have as rigorous as possible description of the business logic of the system, built in the software.

That is why, since the launching of BR approach some efforts to extract BR from existing code are made, but there is no a complete solution of the problem till the moment. Something more, it is not surprising that several authors consider the task unsolvable, and give a solid reasons for their statements (see [3], for example). As usually in the theory of algorithms, if the task is unsolvable or intractable ([4]), one possibility is to solve some particular cases, i.e. to reduce the heavy task to some more simple sub-tasks.

In this paper we try, using the methods of the static analysis of the source code, to split the text of a given program to disjoint parts in such a way, that each part of the partition to be transformed later in a single BR. In Section 2 the necessary definitions and a sample of source code to be used for illustrations are given. Section 3 describes the proposed approach, based on static analysis of the code. The approach is applied to the sample from Section 2 and drafts of corresponding rules are extracted. Section 4 concludes some advantages and disadvantages of the approach, the possibilities to extend the idea to efficient algorithms and describes some intentions for future research.

2. Preliminaries.

2.1. Types of business rules. In [1] the following types of BR are defined:

◇ Definitions of *business terms*. The most basic elements building business rules are terms. So the definitions of terms are business rules which describe how people in business think and talk about things. Terms have been documented in glossaries or as entities in an entity/relationship model.

◇ *Facts* relate terms to each other. The structure of an organization can be described with the facts which relate terms. Facts can be documented as natural language sentences or as relationships, attributes, and generalization structures in a graphical model.

◇ *Constraints*. Each enterprize constrains its behavior in some way. Constraints permit or prevent an action to be taken.

◇ *Derivations*. Business rules of this type define how knowledge in one form have to be transformed into other knowledge, possibly in a different form.

2.2. Static analysis of the code. In this paper we try to apply methods and algorithms for static analysis of the program code. More precisely, we provide a parallel with the algorithms for automated test data generation (ATDG) that we implemented recently [5] as a part of a Smart Source Analyzer [6] – a system, helping software developers of the company. Our test generator uses the path covering method with solving of system of constraints. Traditional stages for such ATDG are:

◇ *Program analysis.* At this stage of ATDG the code is parsed and a control flow graph of the program (CFG) and a data dependence graphs (DDG) for each local variable are constructed. DDG of a variable is used to trace changes of its value during the execution of the program. The stage is typical for almost each task solved with static analysis of the code. We could use this stage of ATDG in extraction of BR without significant changes. Parser of the code in ATDG is created by ANTLR [7] – one of the modern tools for creation of parsers;

◇ *Paths selection.* At this stage of ATDG a set of paths is chosen so as to „cover“ the CFG of the program, following some testing criteria. For extraction of BR the algorithm of this stage have to be changed a bit and so the name of the stage has to be *Path exploring*;

◇ *Generation of test data.* During this stage of ATDG a system (conjunction) of constraints is generated for each of the selected paths. The system is solved by a constraints solver and a test set is constructed. This stage is specific for ATDG and could not be used for extraction of BR. We will replace it with a new stage, adequate to the task, called *Extracting the rules*.

2.3. Sample code. For the consideration below we use the following sample of programming code – simplified version of a real function:

```
public VacReqResult reqVac(Employee anEmployee,            //1
       Date fromDate, Date toDate, VacType type)           //2
{  int daysLeft = anEmployee.getVacDays(type);             //3
   int daysReq = CompCalend.getWorkDays(fromDate,toDate);  //4
   if (daysLeft < daysReq)                                 //5
   {   return VacReqResult.AUTO_REJECT;  }                //6
   else                                                    //7
   {   Employee operLeader = anEmployee.getOperLeader();   //8
       while(operLeader.isOnVac())                         //9
       {   operLeader = operLeader.getOperLeader(); }      //10
       notifyForVacReq(operLeader);                        //11
       return VacReqResult.REQUESTED;                      //12
   }                                                       //13
}                                                          //14
```

Remark. For the purposes of the paper the source code of the sample is too nice (verbose) – all identifiers clearly show the purpose of corresponding type, variable, constant or function/method. In the reality this is not the case. But the mapping of the extracted rules to the ontology of the business domain is going far beyond the scope of this paper.

3. Static analysis approach for BR extraction.

3.1. General limitation. Having in mind that the task for extracting BR from program code could be unsolvable or at least untractable, we have to put some restrictions, changing the task with more easy one. Such restrictions could be, for example, the following:

◇ Let the source code be written in a procedural language, excluding in such a way from consideration the functional and logical programming, as well as the high level languages (e.g. SQL);

◇ Let the program do not contain any form of recursion - neither recursive function calls, both direct and indirect, nor recursively defined user data types or classes of objects;

◇ Let all parameters of function calls be passed *by value*, eliminating in such way the possibility a called function/method to modify some local variables of the calling function;

◇ Let each sequence of characters used as a string, be declared as a `String`, else – as an array of `char`'s; etc.

3.2. Rule extraction principles. Our extraction principles are the following:

◇ Terms and facts will be extracted from the declarative part of the code. The variable names, the constants, the function/method names and the names of the user

defined types/classes are extracted as terms. As a fact is extracted: each declaration that associates a variable with its type, each definition of aggregated type (array, record, union, etc.), as well as each user defined type/class, that associates many variables in the aggregate/type/class; each definition of a function/method or function/method call, which associates the name of the function with its type and the types of its parameters.

◇ Each linear sequence of operators between two branchings (i.e. each part of the program corresponding to an edge of CFG) is extracted as an *elementary derivation*.

◇ And, finally, each branching condition is extracted as two *elementary constraints* – one connected with the true elementary derivation, and one with the false elementary derivation of the branch (the corresponding logical value included).

◇ Elementary derivations and elementary constraints are used for final construction of constraints and derivation rules. This is the heaviest part of the algorithm and we are not ready to specify precisely it now.

3.3. Extracting the rules. Let us now apply the outlined extraction principles to the sample code and to try to point out the main algorithmic challenges of the approach with static analysis of the code.

As it was mentioned above, during the stage of Program analysis the code is parsed and different results will be obtained. First, the CFG of the program shown on the Fig. 1 is obtained. The vertices of CFG are denoted by sets of corresponding code line numbers. Then, the parser determines:

◇ the *user defined types* `VacReqResult`, `Employee`, `VacType`;

◇ the *constants* `AUTO_REJECT`, `REQUESTED` and the *variables* `anEmployee`, `operLeader`, `type`, `daysLeft`, `daysReq`, `fromData`, `toData`;

◇ the *names of functions* `reqVac`, `getVacDays`, `getWorkDays`, `isOnVac`, `getOperLeader`, `notyfyForVacReq`;

which will form our *dictionary of business terms*.

Now we could construct *the facts*. One possible form of fact rule is

<*variable*> `is of type` <*type*>

or the analogical

<*list of variables*> `are of type` <*type*>.

In our example we extract the fact rules

```
f1:  AUTO_REJECT,REQUESTED are of type VacReqResult,
f2:  anEmployee, operLeader are of type Employee,
f3:  fromData, toData are of type Date,
f4:  daysLeft, daysReq are of type int,
```

and

```
f5:  type is of type VacType.
```

Another possible form of fact rule is

<*function name*> `calc` <*type*> `from` <*list of types*>.

Applying the method to an instant of a class of objects we will consider the class as a first argument type in the list of arguments types of the method – in conformance with the syntax of function call.

So, we extract the following facts:

```
f6:  reqVac calc VacReqResult from (Employee, Date, Date,
vacType),
f7:  getVacDays calc int from (Employee, vacType),
```

`f8: getOperLeader calc Employee from (Employee),`
`f9: getWorkDays calc int from (CompCalend, Date, Date),`
`f10: isOnVac calc int from (Employe),`
`f11: notyfyForVacReq calc void from (Employee).`

It is important to stress that for each identified function/method call the extracting algorithm should call itself recursively in order to extract the rules from the code of the called function/method. Among these rules is the derivation rules which describes the transformation/calculation performed by the function/method.

On the second stage the algorithm should explore the CFG of the program in order to outline the branching points – in our example these are the vertices {5,7} and {9}, as well as to extract the *elementary derivations*, corresponding of execution of lines between the start vertex and a branching vertex, between two branching vertices or between a branching vertex and an end vertex, namely {1, 2, 3, 4}, {6}, {8}, {10} and {11, 12}.

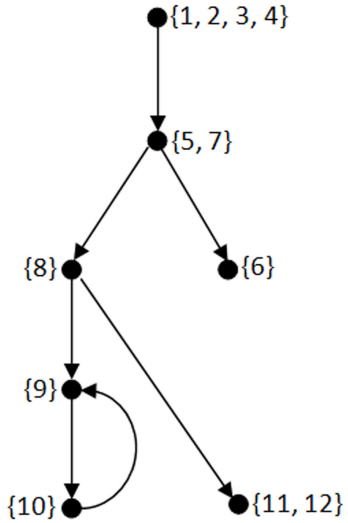

Fig. 1. Control Flow Graph

At the moment we are not able to fix the precise form of the elementary derivations. So we denote them with the general `ederive` <*CFG vertex*>. We obtain the elementary derivations `ed1: ederive {1, 2, 3, 4}`, `ed2: ederive {6}`, `ed3: ederive {8}`, `ed4: ederive {10}` and `ed5: ederive {11, 12}`.

The two conditional expressions in lines {5,7} and {9} generates the following two couples of *elementary constraints*:

`c1t: if (daysLeft < daysReq) is true ed2 and stop,`
`c1f: if (daysLeft < daysReq) is false ed3 and check {9},`
`c2t: if (isOnVac(operLeader)) is true ed3 and check {9},`

and

`c2t: if (isOnVac(operLeader)) is false ed5 and stop.`

The final stage of the algorithm has to compose the *constraints* and *derivation rules*. This is the most challenging part of the extraction process. Different approaches are pos-

sible and all of these approaches have to be investigated further for achieving a satisfiable result.

Here we demonstrate one relatively simple possibility to construct one of the business rules embedded in our sample code. For this purpose we use an element from the algorithm of our ATDG – *covering of a path*. Let us consider the simplest path in CFG of the code leading from the start vertex to one of the end vertices – {1, 2, 3, 4,}, {5,7}, {6}. Covering this path with elements that we constructed to the moment is `ed1`; `c1t` or more precise

`ederive {1, 2, 3, 4}; if (daysLeft < daysReq) is true ederive {6} and stop`.

Now, substitution with the elementary derivations will give us the rule:

Calcuate `daysLeft` *for* (`anEmployee`, `type`) *with* `getVacDay` *and* `daysReq` *for* (`fomDate`, `toDate`) *with* `getWorkDays`. *If* `daysLeft` *is less than* `daysReq` *then issue* `AUTO_REJECT`.

If we are able to map terms into an adequate business ontology, we are able to obtain even more understandable rule formulation.

4. Conclusion. One advantage of this approach is that it is very simple and intuitive – it just follows the syntax of the code. The other advantage is that the implementation of the approach is facilitated by the existence of many software components, used for development of other software tools, based on the static analysis of the code.

There are some elements of the approach that are not fixed precisely yet. First, the set of restrictions that we listed in 3.1 is rather intuitive instead of being result of serious analysis. Each of the included in this list limitations should be an object of detailed research. It could happen that some of these limitations are not necessary or to find an algorithmic way to mitigate them. It is worth to stress that even with these limitations, extracting BRs could be impossible when the developers have been made the code difficult to be read and understood.

The main disadvantage of the approach is that it is not associated with some of the classic models for software development. That could lead to non-consistent set of rules, reflecting the specific style of coding of the involved programmers. An attempt to include some model-oriented part of the approach will give a good combination of simplicity and consistency. It is worth also to study and use elements of another attempts to extract BRs (e.g. [8]) so as to examine their feasibility and efficiency.

REFERENCES

[1] D. Hay, K. A. Healy (eds). Defining Business Rules ∼ What Are They Really? GUIDE Business Rules Project Final Report, rev. 1.3., July, 2000.

[2] B. von Halle. Business Rules Applied, Building Better Systems Using the Business Rules Approach. Wiley Computer Publishing, 2002.

[3] D. Patrick. Caldwell on Softwaree Engineering. `http://dpatrickcaldwell.blogspot.com/search/label/Mainframe Migration`, visited 04.12.2011.

[4] M. R. Garey, D. S. Johnson. Computers and intractability: a guide to the theory of NP-completeness, W. H. Freeman and Company, 1979.

[5] Kr. Manev, A. Zhelyazkov, St. Boychev. Implementation of an Object-Oriented Test Data Generator. Proc. of International Conference on e-Learning and the Knowledge Society – e-Learning'10, Riga, 2010, pp. 231–236.

[6] Smart Source Analyzer (*SSA*), Musala Soft, Ltd.: `http://www.musala.com`
[7] T. Parr. The Definitive Antlr Reference: Building Domain-Specific Languages (1st ed.), Pragmatic Bookshelf, 2007, ISBN 097873925.
[8] I. Baxter, St. Hendrix. A Standards-Based Approach to Extracting Business Rules, Semantic Designs Inc., 2005 (inofficial presentation).

Krassimir Manev
Faculty of Mathematics and Informatics
Sofia University
5, J. Bourchier Blvd.
1164 Sofia, Bulgaria
e-mail: manev@fmi.uni-sofia.bg

Neli Maneva
Institute of Mathematics and Informatics
Bulgarian Academy of Sciences
Acad. G. Bonchev Str., Bl. 8
1113 Sofia, Bulgaria
e-mail: neman@gbg.bg

Haralambi Haralambiev
Musala Soft, Ltd.
36, D. Tzankov Blvd
1057 Sofia, Bulgaria
e-mail: haralambi.haralambiev@musala.com

ИЗВЛИЧАНЕ НА БИЗНЕС ПРАВИЛА ЧРЕЗ СТАТИЧЕН АНАЛИЗ НА ПРОГРАМЕН КОД

Красимир Манев, Нели Манева, Хараламби Хараламбиев

Подходът с използване на бизнес правила (БП) беше въведен в края на миналия век, за да се улесни специфицирането на фирмен софтуер и да може той да задоволи по-добре нуждите на съответния бизнес. Днес повечето от целите на подхода са постигнати. Но усилията, в научно-изследователски и практически аспект, за постигане на „'формална основа за обратно извличане на БП от съществуващи системи“ продължават. В статията е представен подход за извличане на БП от програмен код, базиран на методи за статичен анализ на кода. Посочени са някои предимства и недостатъци на такъв подход.