Modelling of Processes and Formal Methods

D. Riesco, G. Montejano, R. Uzal

Universidad Nacional de San Luis
Departamento de Informática
Ejército de los Andes 950
5700 San Luis - Argentina
driesco@unsl.edu.ar

Abstract

We proposed here a technique, which can be employed within the methodology known as process reengineering. This technique was applied in a government environment, which included an Information System and Geographical Information System, developed with financial support from The World Bank.

One model used in process reengineering is the process model diagram. It allows finding the tasks, to be completed in each area of the organisation. To understand the domain is crucial to be able to specify each one of these tasks.

We show here how to use modelling of processes to find the tasks and to formalise their description using RAISE formal method. In this way, using a model of process as input, an engineer employs a systematic technique to create –as a starting point– the main functions (tasks) of the domain using the RAISE formal method.

Keywords: Reengineering of Processes, Reverse Engineering, Formal Method, RAISE Language

1. Introduction

Business Process Reengineering (BPR) is a method to radically redesign the way in which an organisation performs its tasks [2,3]. BPR is not a method for system development, but a method, which helps in developing a new system, from an old one, which changes the way to do business.

There are two tendencies in the proposed approaches for Process Reengineering [8]. The first of them has been called “white page”. It proposes to ignore the past as a conditioning element and emphasises that nowadays the outstanding quality of a manager is to forget (probably what he has “learnt” will serve for nothing).

Another tendency is the one that includes “Reverse Engineering”, that is, the study of the previous situation to the Process re-planning.

BPR leads to the following changes in:

- Process, the way that the people do the business.
- The software, upgrades of a new system, and how it supports the Process’ Tasks.
- Interface, radical changes in the applied technology.
- Hardware, upgrades to client/server architecture.
- Data, transition to a Relational/Object Oriented Data Base.
- People, change the way the people work, which is affected by the reengineering of processes.

In the implementation of a new system, there are different degrees of changes [7], which show the consequences of processes reengineering:

- Reaffirm: explicit decision to change nothing.
- Repack: change of the user interface (i.e.: move from character interface to GUI).
- Rehost: change of hardware platform (i.e.: migration from centralised environment with mainframe to client/server environment).

- Re-architecture: change of technology kernel without explicitly affecting the way to do the business.
- Reengineering: change the way to do business including all points above.

Therefore, applying process reengineering implies a big change, which affects the way to do business. One of the basic tools applied in reengineering of processes is modelling of process (described in section 2).

The main advantage of modelling of processes is to be able to visualise the way the organisation works, showing the information flow and its different transformations, among different organisation areas.

One of the disadvantages is that the method is informal. Hence, the proposal, presented here, is to combine it with RAISE and obtain all the advantages of a formal method. Section 3 gives a brief description of RAISE formal method, and the proposed technique.

2. Modelling of Processes

Modelling of processes is one of the tools used in process reengineering. This tool allows visualising the processes and flows as well as the organisation areas affected by the processes. It is used conceptually to show what actually happen with the organisation (domain engineering), as well as to be able to visualise what will happen to the new managerial restructuring.

A diagram is created, which represents the areas, departments or sections of the company, showing the processes, data flows and data storages, which belong to each section (or what will happen in the new reorganisation). The tool allows decomposing the diagram of a high level process into other sub processes. The diagram gives a visual representation of the new business activities or those that already exist.

Some key concepts are defined below:

- A Process may be defined as a set of activities that are interlaced or chained. The Process transforms a stimulus born in an external entity to the organisation, into an answer to that external entity. The Process rules the relation of the boundary studied with its environment. In principle, each Process must satisfy one of the components of the demand of the external medium (stimulus generated out of the studied area) that arrive to the organisation area of study.
- Task or Activity is every action conveniently related that constitutes a Process. When a Process is studied, its separation in Tasks constitutes an essential step.
  The enchainment of Tasks transforms a stimulus of the environment in the answer to it. It is important to do the analysis of the value added by each Task to the Process it is part of. The same Task may form part of different Processes. Each Task and may be associated to a job description.
- Separation of Tasks or Activities in Steps. In case of very complex Processes, the simple separation in Tasks is not sufficient to permit an adequate study; each Task should be separated in Steps. A Step can not be separated.
- Formalisation of Steps. Steps are specified using RAISE. They determine the subtasks, which have to be clearly specified to understand the domain.

3. RAISE Formal Method

RAISE [4] is an acronym for “Rigorous Approach to Industrial Software Engineering”. RAISE takes seriously the word “industrial”. The method is intended for use on real development, not just toy examples.

The method is based on a number of principles: separate development, step-wise development, invent and verify, and rigour.

The main point, used in this work, is the concept of separate development. If we want to develop

systems of any size, then we must be able to decompose their description into components and compose the system from the components. Each component can be specified and developed by different engineers. The problem is that an engineer writes a function that others want to use (reuse). It is simple to check the name of the function, which parameters it has and what its result type is. But it is not so easy to be exact about the semantics of the function.

To specify formally the semantic of a function RAISE is used. RAISE´s scheme construct is used to specify each component. RAISE uses a language called RSL (RAISE Specification Language) [5]. In RSL, a scheme (module) is basically a named collection of declarations. A module is a class expression. A class expression represents a class (essentially a set) of models. An object is a named model of a class of models represented by some class expression (the instance of a class).

In a class, we can specify types: build-in types, abstract type (sorts), and compound types (× for Cartesian Product, –set for Set, * for List, $\overrightarrow{m}$ for Maps, Records, Variants).

One of the main aspect of a RAISE specification is the definition of functions (type_expr → type_expr). The specification can be written in a variety of styles: abstract property oriented style (algebraic), model oriented style and concrete algorithm oriented style as an extreme of the spectrum. The properties can be specified as axioms in the class.

4. Formal Specification of the steps specified in the Modelling of Processes.

The technique proposes to write a formal specification in RSL for each step of each task specified in each Process Model.

For that the following points should be kept in mind:

- A Step is specified by a RSL function.
- The functionalities specified in a step have to be absorbed in a RSL scheme.
- As a heuristic, there will be a scheme for each process model data store. This scheme will contain all functions pertaining to the step that access this data store.
- The properties needed to execute the step are specified with function preconditions.
- It is possible to specify other auxiliary functions within the scheme. This helps to understand the step formalisation.
- If it is needed to use functions of other classes (reuse), the scheme will contain a declaration of this class (object declaration).

Above is presented a model applied to the Reengineering of Processes of a State (Province) Land Register environment, where a GIS (Geographical Information System) maintain information about all parcels (lots) of the state.

The step “Unification of Parcels” is very complex to describe unambiguously in natural language; therefore we specify it using RSL formal language. Unification means to create a new parcel with the following properties:

- The unified parcel area is the sum of the area of the parcels, which are unified.
- The building area in the new parcel is the sum of the building area of each unified parcel.
- The adjacent parcels are the same that the adjacent parcels of each unified parcel. The parcels located to the north of the new parcel correspond to the parcels located to north of each unified parcel. The same will occur with the parcels located to the south, east and west.
- Any other properties are also specified using RSL formal language.

Since RSL is used in the steps specification, the RSLTC (RAISE Specification Language Type Checking) tool is employed to type check it. The use of this tool has the advantage of formally checking a specification that has been applied in one model of reengineering of processes.

Here, we present the formal specification of Step "Unification of Parcels":

```
scheme PARCEL =
  class
    type
      Pid,                    /* Parcel Identification */
      P,                      /* Parcel (Abstract Type)*/
      PState,                 /* Parcels of State/Province */
      PP = P × P,
      AdjacentNS = PP-set, /* North-South Adjac.Parcel*/
      AdjacentWE = PP-set, /* West-East Adjac. Parcel */
      Adjacents = AdjacentNS × AdjacentWE
    value                              /* Step Specification */
      unification : P-set × Pid × PState → PState,     /* All adjacent parcels of state */
      adjacents : PState → Adjacents,
      parcels : PState → P-set,                       /* All parcels of the state*/
      adjacentsNS : P × Adjacents → AdjacentNS    /* All adjacents North-South of the parcel */
      adjacentsNS(p,(adjacentNS, adjacentWE)) ≡
            {(p, s) | s : P • (p, s) ∈ adjacentNS} ∪
            {(n, p) | n : P • (n, p) ∈ adjacentNS},
      adjacentsWE : P × Adjacents → AdjacentWE  /* All adjacent West-East of the parcel */
      adjacentsWE(p, (adjacentNS, adjacentWE)) ≡
            {(p, e) | e : P • (p, e) ∈ adjacentWE} ∪
            {(w, p) | w : P • (w, p) ∈ adjacentWE},
      sum : (P → Real) × P-set → Real /* A parcel set is applied a generic function (formal parameter) and sum this result */
      sum(f, ps) as result
        post ps = { } ⇒ result = 0.0 ∧
             ps ≠ { } ⇒ let p : P • p ∈ ps in
               result = sum(f, ps \ {p}) + f(p)
               end,
      id : P → Pid,    /* Functions about Parcel atributes */
      groundArea : P → Real,
      buildingArea : P → Real
    axiom
      ∀ ps : P-set, pid : Pid, pState : PState •
        unification(ps, pid, pState) as pStateR
        post ∃p': P • id(p') = pid ∧
          groundArea(p') = sum(groundArea, ps) ∧
          buildingArea(p') = sum(buildingArea, ps) ∧
          /* Adjacent Parcels locate to N-S of new parcel*/
          adjacentsNS(p', adjacents(pState)) =
            {(n, p) | n : P, p : P • p ∈ ps ∧
            (n, p) ∈ adjacentsNS(p, adjacents(pState)) ∧
            n ∉ ps} ∪
            {(p, s) | s : P, p : P • p ∈ ps ∧
            (p, s) ∈ adjacentsNS(p, adjacents(pState)) ∧
            s ∉ ps} ∧
          /*Adjacent Parcels locate to W-E of new parcel*/
          adjacentsWE(p', adjacents(pState)) =
            {(w, p) | w : P, p : P • p ∈ ps ∧
            (w, p) ∈ adjacentsWE(p, adjacents(pState)) ∧ w ∉ ps} ∪
            {(p, e) | e : P, p : P • p ∈ ps ∧
            (p, e) ∈ adjacentsWE(p, adjacents(pState)) ∧
            e ∉ ps} ∧
              /* Rest of the parcels keep same adjacents */
```

```
        ∃ p : P • p ≠ p' ∧
          p ∈ parcels(pState) ∧ p ∉ ps ⇒
            adjacentsNS(p, adjacents(pState)) =
            adjacentsNS(p, adjacents(pState))
end
```

5. Conclusions

The technique presented here has all the advantages of the use of formal methods in the first step of the reengineering of processes. We apply the modelling of processes and specify each task of it using a formal specification, in this case using the RAISE formal method.

The technique was applied in a government environment using the Modelling of Processes of Oracle Designer 2000, and the RSLTC (RAISE Specification Language Type Checking) to specify each task described with the Oracle tool.

This technique allows the engineering group, which will do the forward engineering, to have a clear, unambiguous specification of each task done by the organisation. In this way the problem of vagueness inherent in an informal description is avoided which helps to construct a more reliable system. It also facilitates the rigorous specification and analysis of complex software systems.

The RAISE specification can be a contract between the developers and the users, as well as between the developers, in charge of specifying the model of the processes, and the developers, in charge of the forward engineering.

References

[1] Jacobson, I. and others “Object Oriented Software Engineering. A use Case Driven Approach”, Reading MA Addison Wesley, 1992

[2] Jacobson, I. “Objectifying Business Process Reengineering”, Addison Wesley, 1996

[3] Jacobson, I. and Lindström F., “Re – engineering of old systems to an object – oriented architecture”, OOSPLA´91, pp. 340 – 350

[4] The RAISE Method Group, "The RAISE Development Method", Prentice Hall, 1995.

[5] The RAISE Language Group, "The RAISE Specification Language", Prentice Hall, 1992.

[6] Hammer, M. and Champy, J. “Reengineering the Corporation: A Manifesto for Business Revolution”, Harper Collins Publishing, Inc., 1993

[7] Champy, J. “Reengineering Management”, HarperBusiness, 1995

[8] Manganelli, R. and Klein, M. “The Reengineering Handbook”, AMACON, 1994

[9] Lincoln, T., “Managing Information System for Profit”, John Wiley, 1990

[10] Martin, J. “Information Engineering”,PrenticeHall,90

[11] Mc Cabe T. J. “A Software Complexity Measure”, IEEE Trans. Software Engineering, vol 2, no. 6, Dec76

[12] Arthur, Lowell Jay, “Rapid Evolutionary Development”, Wiley, 1992

[13] Rockart, John F., “Chief executives define their own data needs”, Harvard Business Review, March – April, 79