Using Business Process Reengineering to obtain a RAISE Specification

Daniel E. RIESCO, Germán A. MONTEJANO, Roberto UZAL
Universidad Nacional de San Luis
Departamento de Informática
Ejército de los Andes 950
5700 San Luis - Argentina
{driesco| gmonte| ruzal}@unsl.edu.ar

ABSTRACT

We have proposed a technique which can be employed within the methodology known as business process reengineering. This technique has been applied in a government project, 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, which helps to find 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 in this work how to use process modelling 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.

Furthermore, we show how a structured architecture in layers can be used for reuse during the development in the large.

Keywords: Process Reengineering, Process Modelling, Formal Method, RAISE Method, RSL Language

1. INTRODUCTION

Business Process Reengineering (BPR) is a method to radically redesign the way in which an organisation performs its tasks [1,2]. 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 [3]. 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’s 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 process reengineering.

In the implementation of a new system, there are different degrees of changes [4], which show the consequences of process 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 to process reengineering is process modelling (described in section 2).

The main advantage of process modelling 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 following sections explain the proposed technique.

2. PROCESS MODELLING

Process modelling [5] 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 happens with the organisation (domain engineering), as well as to be able to visualise what will happen to the new managerial restructuring.

A diagram [6] 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 [7] 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) [8]. 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 ($\times$ for Cartesian Product, –set for Set, * for List, $\xrightarrow[m]{}$ for Maps, Records, Variants).

One of the main aspect of a RAISE specification is the definition of functions (type_expr $\rightarrow$ 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 PROCESS MODELLING.

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 belonging 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).

The steps appear from a model applied to the Process Reengineering 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 Checker) tool is employed to check it. The use of this tool has the advantage of formally checking a specification that has been applied in one model of process reengineering.

Each scheme surged from the model is structured to conform the architecture of business domain.

In the next sections, we present the formal specification based on layer architecture.

5. ARCHITECTURE

After we describe the process model and specify each step, we build the architecture. The architecture is composed of three layers: specific layer, general layer and middleware layer [9].

A layer is a set of schemes that share the same degree of generality. Lower layers are general to several domain specifications, while higher layers are more specific to a concrete domain. A specific scheme, which is located in the specific layer, can use general layer or middleware layer schemes. The scheme, which is located in general layer, can use schemes of middleware layer. The layers are hierarchical.

The schemes defined in the general layer, are general at this kind of business, therefore they are only useful in this domain and they can not be used in others domains. The schemes defined in the middleware layer are so general that they can be used in any domain. Examples of middleware layer are standard specifications as bags, stacks, queues, etc. in different abstract specification (applicative sequential, imperative sequential, imperative concurrent).

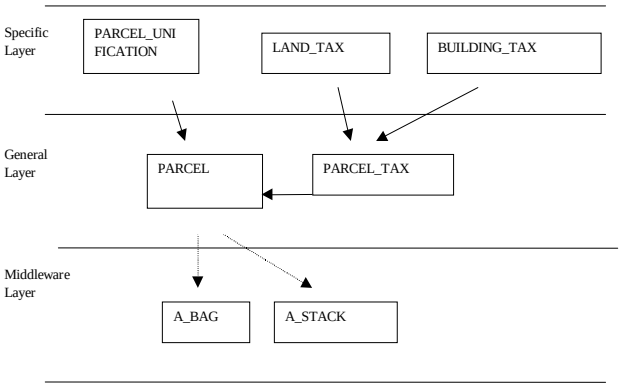

Figure 1: Scheme layers of the Process "Unify Parcels"

Figure 1 shows some schemes of the "Unify Parcels" process and the architecture layers.

This architecture helps to understand the formalization of the process model. If it is needed to use functions of other schemes in the same or in different layer, the scheme will contain a declaration of these schemes (factoring step).

6. FORMALIZING THE "UNIFY PARCELS" PROCESS

The main steps of the "Unify Parcels" Process were formalized. They are "Register the Unification of Parcels", “Calculate land tax” and “Calculate build tax”.

In the following sections, the formalization of main steps of the Process Model is presented. First, the general layer schemes are showed: PARCEL and PARCEL TAX, and then the specific layer schemes: PARCEL_UNIFICATION, LAND_TAX and BUILDING_TAX.

6.1 Parcel

This scheme is generic for the domain. It is in the general layer. All common functions among all steps are factorized in one scheme called PARCEL.

scheme PARCEL = **class**
type
  Pid, /* Parcel Identification */
  P, /* Parcel (Abstract Type) */
  PState, /* Parcels of State/Province */
  PP = P × P,

  /*North-South Adjacent Parcel */
  AdjacentNS = PP-**set**,

  /* West-East Adjacent Parcel */
  AdjacentWE = PP-**set**,

  Adjacents = AdjacentNS × AdjacentWE

value
  /* All adjacent parcels of the state */
  adjacents : PState → Adjacents,

  /* All parcels of the state */
  parcels : PState → P-**set**,

  /* All adjacents North-South of the parcel */
  adjacentsNS : P × Adjacents → AdjacentNS
  adjacentsNS(p, (AdjacentNS, AdjacentWE)) ≡
    {(p, s) | s : P • (p, s) ∈ adjacentNS}
    ∪
    {(n, p) | n : P • (n, p) ∈ adjacentNS},

  /* All adjacent West-East of the parcel */
  adjacentsWE : P × Adjacents →
    AdjacentWE
  adjacentsWE(p, (AdjacentNS,
    AdjacentWE)) ≡
    {(p, e) | e : P • (p, e) ∈ adjacentWE}
    ∪
    {(w, p) | w : P • (w, p) ∈ adjacentWE},

  /* A parcel set is applied a function(formal parameter) generic and sum this result */
  sum : (P → **Real**) × P-**set** → **Real**
  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,

  /* Functions about Parcel attributes */
  id : P → Pid,
  groundArea : P → **Real**,
  buildingArea : P → **Real**

end

6.2 Parcel Tax

This scheme factorizes the common functions of calculation of land tax and build tax. These functions are generic for different concrete applications but are common for the same domain.

```
PARCEL    /* Reference to use this scheme */

scheme  PARCEL_TAX =
        extend PARCEL with class

type
/* Basic Value by block and by square meter*/
  BV

value
  front : P → Real,       /* Parcel Front */
  bottom : P → Real,    /* Parcel Bottom */
  /* Basic Value by block and by square meter
     according to the location */
  bVal : P × PState → Real

end
```

6.3 Step Register Parcels Unification

Unification means to create a new parcel with the following properties: the unified parcel area is the sum of the area of the unified parcels and the new building area is the sum of the unified parcels building areas.

The adjacent parcels are the union of 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 property is also specified using RSL formal language.

The formal specification of Activity "Register Parcels Unification" is presented.

```
PARCEL   /* Reference to use this scheme */

scheme  PARCEL_UNIFICATION =
        extend PARCEL with class

value
  /* Step Specification */
  unification : P-set × Pid × PState →  PState

  axiom
  ∀ ps : P-set, pid : Pid, pState : PState •
    unification(ps, pid, pState) as pStateR post
    let p' : P in
      id(p') = pid
      ∧ groundArea(p') = sum(groundArea, ps)
      ∧ buildingArea(p') = sum(buildingArea,
ps)
      ∧
/* Adjacent Parcels locate to North-South of
the 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 West-East of the
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 parcels keep the same adjacents */
      let p : P in
        p ≠ p' ∧
        p ∈ parcels(pState) ∧
        p ∉ ps ⇒
          adjacentsNS(p, adjacents(pState)) =
          adjacentsNS(p, adjacents(pState))
      end
    end
end
```

6.4 Land Tax

Here, we formalize the step "Calculate land tax". It is an extension of the PARCEL_TAX, which is in turn an extension of PARCEL.

```
PARCEL_TAX    /* Reference to use this
scheme */

scheme  LAND_TAX =
        extend PARCEL_TAX with class

type
/* Coefficient of adjustment of basic value
according the relationship between the parcel
front and bottom or between area and bottom
*/
  Co = Real × Real ─m→ Real,
  Shape ==        /* The ground shape is */
```

```
/* Regular with less than 2000 square meters
and it is not in corner of block */
    RG_l2k_NCrn |
/* Irregular with less than 2000 square meters
and it is no in corner of block */
    IRG_l2k_NCrn

value
/* Coefficient of adjustment of basic value
according to the relationship between the
parcel front and bottom, when the parcel has
the front on only street and the area is less
than 2000 square meters */
  co1 : Co,

/* Coefficient of adjustment of basic value
according to relationship between the front
size and the parcel area */
  co2 : Co,

/* and so on with all relationships co3, co4, */

/* Co return the coefficient of adjustment */
  co : P × Co × Real × Real → Real,
  shape : P → Shape,

  pVal : P × PState → Real
  pVal(p, pSt) ≡
    case shape(p) of
      RG_l2k_NCrn → bVal(p, pSt) *
          co(p, co1, front(p), bottom(p)) *
          groundArea(p),
      IRG_l2k_NCrn → (groundArea(p) /
front(p)) *
          bVal(p, pSt) * groundArea(p)

/* SO ON ... (to be completed with 24 cases)*/

   end
end
```

6.5 Build Tax

This scheme formalizes the step "Register build tax". This scheme is an extension of the PARCEL_TAX.

```
PARCEL_TAX    /* Reference to use this
scheme */

scheme  BUILDING_TAX =
          extend PARCEL_TAX with class
type
/* Categories of the Building */
  BCo = Age × State × Ct →m Real,
  Ct,              /* Category */
  Age,             /* Building Age */
  State            /* State of the Building */

value
  bCo : BCo,
  ct : P → Ct,              /* Parcel Category */
  state : P → State,
  bCo : P → Real,
  pBVal : P × PState → Real
  pBVal(p, pSt) ≡
    bVal(p, pSt) * bCo(p) * buildingArea(p)

/* SO ON ... */

end
```

7. CONCLUSIONS

The technique presented here has all the advantages of the use of formal methods in the first step of the process reengineering. We apply the process modelling 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 project using the Process Modelling of Oracle Designer 2000, and the RSLTC (RAISE Specification Language Type Checker) to specify each task described with the Oracle tool.

This technique allows the engineers group, who 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 used as a contract between the developers and the users. It can be used between developers who specify the process model and the developers who do the forward engineering as well.

In this work we show the use of an architecture divided in three layers. The power of this proposal resides in the reuse of scheme specifications inside the same domain for different developments.

8. REFERENCES

[1] Hammer, M. and Champy, J. “Reengineering the Corporation: A Manifesto for Business Revolution”, Harper Collins Publishing, Inc., 1993
[2] Jacobson, I. “Objectifying Business Process Reengineering”, Addison Wesley, 1996
[3] Manganelli, R. and Klein, M. “The Reengineering Handbook”, AMACON, 1994
[4] Champy, J. “Reengineering Management”, HarperBusiness, 1995
[5] Jacobson, I. and others “Object Oriented Software Engineering. A use Case Driven Approach”, Reading MA Addison Wesley, 1992
[6] M. Aguilar, T. Rautert, and A. Pater, "Business Process Simulation: A Fundamental Step Supporting Process Centered Management", Proc. IEEE, Winter Simulation Conference, 1999, pp.1383-1392.
[7] The RAISE Method Group, "The RAISE Development Method", Prentice Hall, 1995.
[8] The RAISE Language Group, "The RAISE Specification Language", Prentice Hall, 1992.
[9] Jacobson, I., Booch, G., and Rumbaugh, J., "RUP (Rational Unified Process) 2002", www.rational.com/rup.