Pseudocode

1 Introduction

Within BCIT, a business activity, compliance process, compliance requirement or an IT component can be changed. In the following, we show the pseudocodes for each changed element to analyze the interaction between compliance and the change patterns ‘delete element and ‘replace element’. In addition, we present pseudocode to query for alternative compliance processes and their integration into the business process.

2 Change of a Business Activity

Algorithm: determine the interaction by deleting a business acivity

Input: Graph g, element that shall be removed v ∈ g where h(v)=business process

```
1  // get all obsolete compliance requirements
2  obsolete.add(v)
3  Foreach cr in (get all predecessor of v where h=compliance requirement) do
4  |  dir_suc = get direct successor of cr
5  |  If (dir_suc are only part of obsolete) do
6  |  |  obsolete.add(cr)
7  |  |  Foreach cp in (get direct predecessor of cr where h=compliance process) do
8  |  |  |  satisfied_compliance = get direct successor of cp where h=compliance requirement
9  |  |  |  If (satisfied_compliance are only part of obsolete) do
10 |  |  |  |  obsolete.add(cp)
11 |  |  |  |  obsolete.add(determine the interaction by deleting a compliance process(cp))
12
13 //get all obsolete compliance requirements by IT
14 Foreach IT in (get all predecessor of v where h=IT component) do
15 |  dir_suc = get direct successor of it
16 |  If (dir_suc are only part of obsolete) do
17 |  |  obsolete_it.add(IT)
18 Foreach IT in (obsolete_it) do
19 |  Foreach compliance in (get all predecessor of IT where h=compliance requirement) do
20 |  |  If (get direct succesor(compliance) are only part of obsolete or obsolete_it) do
21 |  |  |  obsolete.add(IT, compliance)
22 generate result_graph based on g and obsolete
```

Output: Graph result_graph

Algorithm: determine the interaction by replacing a business acivity

Input: Graph g, element that shall be replaced v ∈ g where h(v)=business process

```
1  // get all direct demanding compliance requirements
2  Foreach cr in (get all predecessors of v where h=compliance requirement) do
3  |  mark cr as direct demanding AND add to result
4  |  get compliance process of cr and add to result
5
6  // get all indirect demanding compliance requirements
7  Foreach IT in (get all predecessor of v where h=IT component) do
8  |  Foreach cr in (get all predecessor of IT where h=compliance requirement) do
9  |  |  mark cr and IT as indirect demanding and add to result
10 generate result_graph based on g and result
```

Output: Graph result_graph

3 Change of a Compliance Process

Algorithm: determine the interaction by deleting a compliance process

Input: Graph g, element that shall be deleted v ∈ g where h(v)=compliance process

```
// get all violated compliance requirements
Foreach cr in (get all direct successor of v where h=compliance requirement) do
    Foreach cr_2 in (get all predecessor of cr where h=compliance requirement) do
        mark cr and cr_2 as violated AND add to result

// get all obsolete compliance requirements
obsolete_IT=[];
Foreach IT in (get all direct predecessor of v where h=IT component) do
    If (IT has only one direct successor)
        obsolete_IT.add(IT)
        Foreach IT_pred in (get all predecessor of IT) do
            If (IT_pred has only successors that are part of obsolete_IT) do
                obsolete_IT.add(IT_pred)
Foreach IT in(obsolete_IT) do
    Foreach cr (get all predecessor of IT where h=compliance requirement) do
        Foreach leaf (get all leaf of cr where h=compliance requirement) do
            If (direct successor of cr are only part of obsolete_IT) do
                mark cr, leaf, getNodesBetween(cr, leaf) and obsolete_IT as obsolete and add to result
generate result_graph based on g and result
```

Output: Graph *result_graph*

Algorithm: determine the interaction by replacing a compliance process

Input: Graph g, element that shall be replaced v ∈ g where h(v)=compliance process

```
// get all direct demanding compliance requirements
Foreach cr in (get all direct successor of v where h=compliance requirement) do
    Foreach cr_2 in (get all predecessor of cr where h=compliance requirement) do
        mark cr and cr_2 as direct demanding AND add to result

// get all indirect demanding compliance requirements
Foreach IT in (get all predecessor of v where h=IT component) do
    Foreach cr in (get all predecessor of IT where h=compliance requirement) do
        mark cr and IT as indirect demanding and add to result
generate result_graph based on g and result
```

Output: Graph *result_graph*

4 Change of a Compliance Requirement

Algorithm: determine the interaction by deleting a compliance requirement

Input: Graph g, element that shall be removed v ∈ g where h(v)=compliance requirement

```
// get all obsolete compliance requirements
obsolete.add(v)
Foreach cr in(get all successor of v where h=compliance requirement) do
    dir_pred = get direct precedessor of cr where h=compliance process
    If(dir_pred are only part of obsolete) do
        obsolete.add(cr)

Foreach cr in (get all predecessor of v where h=compliance requirement) do
    dir_suc = get direct successor of cr where h=compliance process
    If(dir_suc are only part of obsolete) do
        obsolete.add(cr)

// get all obsolete Compliance Processes
Foreach cr in(obsolete) do
    Foreach cp in (get direct predecessor of cr where h=compliance process) do
        suc = get direct successor of cp where h=compliance requirement
        If(suc are only part of obsolete) do
            obsolete.add(cp)
            obsolete.add(determine the interaction by deleting a compliance process(cp))
generate result_graph based on g and obsolete
```

Output: Graph *result_graph*

Algorithm: determine the interaction by replacing a compliance requirement

Input: Graph g, element that shall be replaced v ∈ g where h(v)=compliance requirement

```
1  // get all direct demanding compliance requirements
2  Foreach cr in (get all predecessors of v where h=compliance requirement) do
3  └ mark cr as direct demanding AND add to result
4
5  // get all indirect demanding compliance requirements
6  Foreach cr in (get all sucessors of v where h=compliance requirement) do
7  │ mark cr as indirect demanding AND add to result
8  │   Foreach cp in (get all direct predecessor of v where h=compliance process) do
9  └   └  mark cp as indirect demanding AND add to result
10 generate result_graph based on g and result
```

Output: Graph *result_graph*

5 Change of a IT Component

Algorithm: determine the interaction by deleting an IT component

Input: Graph g, element that shall be deleted v ∈ g where h(v)=it architecture

```
1  // get all violated compliance requirements
2  Foreach it in (get all leafs of v where h=it architecture) do
3  │ Foreach complianceprocess in (get all direct successor of it where h=compliance process) do
4  │ │ Foreach cr in (get all direct successor of complianceprocess where h=compliance requirement) do
5  │ │ │ i = get all predecessor of cr
6  └ └ └ mark cr and i as violated AND add to result
7  // get all obsolete compliance requirements
8  add v to list_it
9  Foreach it in (get all successor of v where h=it architecture) do
10 │ If (it only requires v)
11 └ └ add it to list_it
12 Foreach it in (list_it) do
13 │ Foreach cr in (get all direct predecessor of it where h=compliance requirement) do
14 │ │ If (get all direct successor of cr==it)
15 │ │ │ mark cr as obsolte AND add to result
16 │ │ │ Foreach cr2 in (get all predecessor of cr where h=compliance requirement) do
17 │ │ │ │ If (cr2 hasn't other direct successor than get all predecessor of cr where h=comp.req.)
18 └ └ └ └ └ mark cr2 as obsolte AND add to result
19 generate result_graph based on g and result
```

Output: Graph *result_graph*

Algorithm: determine the interaction by replacing an IT component

Input: Graph g, element that shall be replaced v ∈ g where h(v)=it architecture

```
1  // get all direct related compliance requirements and compliance processes to v
2  Foreach i in (get all predecessor of v where h=compliance requirement) do
3  └ mark i as direct AND add i including all vertices between i und v to result
4  // get all transitive related compliance processes and compliance requirements to v
5  Foreach it in (get all leafs of v where h(v)=it architecture) do
6  │ Foreach activity in (get all direct successor of it where h=business process) do
7  │ │ Foreach cr in (get all direct predecessor of activity where h=compliance requirement) do
8  │ │ │ i = get all predecessor of cr
9  │ └ └ mark it, activity, cr and i as transitiv AND add to result
10 │ Foreach complianceprocess in (get all direct successor of it where h=compliance process) do
11 │ │ Foreach cr in (get all direct successor of complianceprocess where h=compliance requirement) do
12 │ │ │ i = get all predecessor of cr
13 └ └ └ mark it, complianceprocess, cr and i as transitiv AND add to result
14 generate result_graph based on g and result
```

Output: Graph *result_graph*

6 Query alternative Compliance Process and propose compliant Business Process

Algorithm: Query and Integrate alternative Compliance Processes

Input: alternative graph *ag*, result graph *rg*, process graph *pg*

```
1  // get all violated compliance processes
2  Foreach cp in (get all violated compliance processes in rg) do
3  |  // get all alternative compliance processes
4  |  Foreach sibling_cp in (get sibling compliance processes of cp) do
5  |  |  // check the executability of alternative compliance process
6  |  |  req_it = silbing_cp.getRequiredIT
7  |  |  If (req_it OR req_it.predecessor not in rg)
8  |  |  |  add sibling_cp to alternatives
9  |  // search for compliance process patterns
10 |  If (alternatives is empty)
11 |  |  add get_all_succesor(cp, type(suc_cp)=compliance process pattern) to alternatives
12 // integrate alternative cp into the business process graph pg
13 Foreach element in alternatives do
14 |  bp = pg
15 |  removeViolatedComplianceProcess from bp
16 |  integrate element in bp
17 |  add bp to compliantBusinessProcesses
```

Output: list process_graph *compliantBusinessProcesses*