Appendix 1: Algorithms

Algorithm A-1: Identifying and summarizing the sequence pattern

Sequential-Pattern (Matrix A, Qc)
  set Flag-seq to zero
  while (1)
    Set Flag-Seq to zero
    ∀ Cur_node in Matrix
      if **Relation (13)** is true between Cur_node and Next_node:
      Call Cal_Perform_Sequential (A, Cur_node, Next_node, Qc)
        set Flag-Seq to 1
      end
    end
  end
  if Flag-Seq is zero
    Break the loop
  end
End
Function Cal_Perform_Sequential (Matrix A, Cur_node, Next_node, Qc)
  $Availability_{Cur\_node}$, $ResponseTime_{Cur\_node}$, $Cost_{Cur\_node}$=$Qc_{1:3}$(Cur_node)
  $Availability_{Next\_node}$, $ResponseTime_{Next\_node}$, $Cost_{Next\_node}$=$Qc_{1:3}$(Next_node)
  Set $Probability_{Cur\_node} = Probability_{Next\_node}$
  Update the Matrix A (Remove all of connection of Next_node to other and vice versa)
  Set $Availability_{Cur\_node} = Availability_{Cur\_node} \times Availability_{Next\_node}$
  Set $ResponseTime_{Cur\_node} = ResponseTime_{Cur\_node} + ResponseTime_{Next\_node}$
  Set $Cost_{Cur\_node} = Cost_{Cur\_node} + Cost_{Next\_node}$
  $Qc_{1:3}$(Cur_node) = $Availability_{Cur\_node}$, $ResponseTime_{Cur\_node}$, $Cost_{Cur\_node}$
  Delete $Qc_{1:3}$(Next_node)
End

Algorithm A-2: Identifying and summarizing the parallel pattern

Function Parallel-Pattern (Matrix A, Qc)
  set Flag-Paral to zero
  while (1)
    ∀ Cur_node in Matrix A
      if Cur_node has the Concurrent_node and the **Relation (14)** or **Relation (18)** is true between them
        Call Cal_Perform_Parallel (A, Cur_node, Concurrent_node, Qc)
        Set Flag-Paral to 1
      end
    end
  end
  if Flag-Paral is zero
    Break the loop
  end
End

Cal_Perform_Parallel (Matrix A, Cur_node, Concurrent_node, Qc)
  $Availability_{Cur\_node}$, $ResponseTime_{Cur\_node}$, $Cost_{Cur\_node}$=$Qc_{1:3}$ (Cur_node)
  $Availability_{Concurrent\_node}$, $ResponseTime_{Concurrent\_node}$, $Cost_{Concurrent\_node}$=$Qc_{1:3}$(Concurrent_node)
  Set $Probability_{Cur\_node} = 1$
  Update the Matrix A (Remove all of connection of Concurrent_node to other and vice versa)
  Set $Availability_{Cur\_node} = Availability_{Cur\_node} \times Availability_{Concurrent\_node}$
  Set $ResponseTime_{Cur\_node}$=Max $\{ResponseTime_{Cur\_node}, ResponseTime_{Concurrent\_node}\}$
  Set $Cost_{Cur\_node} = Cost_{Cur\_node} + Cost_{Concurrent\_node}$
  $Qc_{1:3}$(Cur_node) = $Availability_{Cur\_node}$, $ResponseTime_{Cur\_node}$, $Cost_{Cur\_node}$
  Delete $Qc_{1:3}$(Concurrent_node)
End

Algorithm A-3: summarizing the loop pattern

1. ***Function Loop_Pattern*** *(Matrix A, Qc)*
2. *set Flag-Loop to zero*
3. *Loop_array=**Call Identifying_Loop_Pattern** (Matrix A)*
4. *if (Loop_array! =null)*
5. *set Flag-Loop to one*
6. *for i=1: length (Loop_array)*
7. *Set Array_node =Loop_array(i)*
8. *if **Relation (17)** or **Relation (20)** or **Relation (21)** are true in Array_node:*
9. ***Call Cal_Perform_Loop (A, Array_node, Qc)***
10. *end*
11. *end*
12. *end*
13. ***End***

14. ***Function Cal_Perform_Loop*** *(Matrix A, Array_node, Qc)*
15. $Availability_{Array\_node}$, $ResponseTime_{Array\_node}$, $Cost_{Array\_node}$ *=Qc (Array_node)*
16. *Probability of every node in loop calculated by formula in Table 2*
17. *Update the Matrix A (Create connections from the v_1 to the external nodes of loop and remove the connections of $v_2, \ldots, v_n$ to the external nodes of loop)*
18. *Set $Availability_{Array\_node}$ by formula in Table 1*
19. *Set $ResponseTime_{Array\_node}$ by formula in Table 1*
20. *Set $Cost_{Array\_node}$ by formula in Table 1*
21. $Qc_{1:3}(Array - node(1)) = Availability_{Array\_node}$, $ResponseTime_{Array\_node}$, $Cost_{Array\_node}$
22. *Delete $Qc_{1:3}$(Array_node (2: length (Array_node))*
23. ***End***

Algorithm A-4: Identifying the loop pattern

1. ***Loop_array =Function Identifying_Loop_Pattern (Matrix A)***
2. *Add first node of graph to stack*
3. *set Visited-array to empty*
4. *while (1)*
5. *if the stack is empty*
6. *Break;*
7. *end*
8. *if the last member of stack is in visited array*
9. *Remove last member from the end of stack*
10. *Pos_last =find position last member of stack in visited array*
11. *∀ Cur_node in visited array from Pos_last position to the end of visited array*
12. *if Cur_node has connection with the last member of stack*
13. *Loop= [last member, …, Cur_node]*
14. *Add Loop to Loop_array*
15. *end*
16. *else if last member has at least one child*
17. *Add last member to visited array*
18. *Remove last member of end of stack*
19. *Add last member's children to the end of stack*
20. *else*
21. *Break*
22. *end*
23. *end*
24. *Sort the loops in Loop_array in order from small to large*
25. *Return Loop_array*
26. ***End***

Algorithm A-5: Extracting paths from the summarized composition

1. ***[Npath, Path] =ExtractingPath-Graph (Matrix A, Ipath)*** */Path is a structure of arrays whose each member contains web services of a graph path*
2. *i=1;*
3. *array (1) =Start_node /* put the graph Start_node in the first place of the array; array is a structure that contain a number of arrays*
4. *while (~isempty (array))*
5. *If (j has children) /* j=the latest node of array*
6. *find the children of j*
7. *for each of the children of j /* we denote the children of j by k*
    a. *create vector;*
    b. *vector= [array (1), k]*
    c. *Extend the array by adding the vector to the array /* we will have array (1), array (2), …array (Ipath)}*
8. *else*
9. *Path(i)=array (1)*
10. *i=i+1;*
11. *end*
12. *delete the array (1)*
13. *end /* end while*
14. *Npath=length (Path) /* Npath is the number of paths in summarized graph*
15. *Return (Path (Ipath), Npath)*
16. ***End***