BPEL 流程监视的可视化实现①

孙　嵴　叶世阳　魏　峻　(中国科学院软件研究所 软件工程研究开发中心 北京 100190)

Implementation of Visual BPEL Process Monitoring

Zheng Sun, Shiyang Ye, Jun Wei (Institute of Software, Chinese Academy of Sciences, Beijing 100190)

Abstract: When a BPEL process is executed, it is necessary to dynamically monitor the process. BPEL is a executable language, which is not suitable for visual monitoring. On the other hand, BPMN is designed to visually describe business process and is more intuitive for monitoring. To visually monitor a BPEL process, transformation from BPEL to BPMN is necessary. However, current study of transformation from BPEL to BPMN does not support the transformation of "link" activity. Besides, no work has been done to add supplementary information into BPMN during transformation. In this paper, we transform nested BPEL process into a flat BPMN process graph without hierarchy through applying a flattening strategy. Especially, we analyze various scenarios of the transformation of link activity, and provide a method to deal with it. Besides, we analyze the mapping between BPEL activities and BPMN graph, through which we found out that some supplementary information cannot automatically obtained from BPEL process. These supplementary information need to be added during transformation. At the end of this paper, we present the structure of our monitoring tool which is based on our transformation algorithm.

Key words: BPEL; BPMN; flatrening strategy; link acivity

1　Introduction

With the maturation of SOA and Business Process Manage technology, Process-Oriented develop is increasingly being used in a wide range of applications. WS-BPEL[1] is an important specification for composing services in services computing. BPEL composes services into complex business process. When a BPEL process is executed, there is a need to dynamically monitor the process as well as present the monitoring information intuitively in the form of process diagram. However, BPEL is an executable language, with a low abstraction level, which is not suitable for visual monitoring. BPMN[2] is a language designed to visually describe business process, by International Standard Organization BPMI. It specifies a set of standard graphical notations in a high abstraction level and is more intuitive than BPEL. Besides, one goal of BPMN is to visually depict business process execution language, such as BPEL with business-oriented notations[2]. Therefore, BPMN is more suitable for visual process monitoring and it is a good idea to transform BPEL to BPMN when implementing the visual BPEL monitoring tool.

However, the transformation does not support all the complex structures from BPEL to BPMN well, such as the link structure. In addition, there is no precise analysis of the mappings between BPEL activities and BPMN annotations. Because these mappings are not one-to-one and there is information in BPMN that is not contained in BPEL, such as the annotation coordination and size. More work must be done while transforming from BPEL

① Supported by the National Grand Fundamental Research 973 Program of China under Grant No.2009CB3070; National High-Tech Research and Development Plan of China under Grant No.2007AA010301

to BPMN.

In order to solve these problems, we design a transformation algorithm. By applying a top-down flatting strategy, we can flatten the nested BPEL process flow into a flat process graph without hierarchy. Especially, we discuss possible scenarios of the transformation of "link" activity, and provide an algorithm to deal with it. Meanwhile, we analyze the mapping between BPEL activities and BPMN annotations, and add essential supplementary information into BPMN process diagram during the course of transformation.

Finally, we implement the transformation algorithm in our visual BPEL process monitoring tool, which is capable of transforming BPEL to BPMN and visually monitoring business process execution.

The rest of the paper is organized as follows. Section 2 discusses the related work. In Section 3, we discuss the transformation from BPEL to BPMN, in which, we focus on the flatting algorithm and link activity transformation. Besides, we analyze the mapping between BPEL activities and BPMN annotations, and method to add supplementary information to BPMN. Section 4 introduces our visual BPEL monitor tool, and finally Section 5 concludes our paper.

2 Related Work

There has been much work on the transformation from BPMN to BPEL, however little has been done from BPEL to BPMN. Although this transformation has been done by Together -a model building tool by Cor. Borland and STPBPMN-a open source project, they failed in integrity of the transformation and lacked the support for structure "link". Besides, the BPMN, got from BPEL, keeps nested structure, which is disadvantageous for analysis and communication.

Jan Recker and Jan Mendlings in Refs.[3,4] proposed three strategies for transformation from Block-Oriented modeling language to Graph-Oriented modeling language, including Flattening, Hierarchy-Preservation, and Hierarchy-Maximization. These strategies are proposed for general transformation from block structure to graph structure, not specific to any process modeling language. Thus, no special mapping from BPEL activities to BPMN elements is discussed. In this paper, we apply and improve the flattening strategy in the transformation from BPEL to BPMN and make it the foundation of our algorithm.

3 The Translation from BPEL to BPMN

The primary goal of BPMN is to provide a notation that is readily understandable by all business users [1]. BPMN based on Directed Graphs and has visual appearance. BPEL focuses on the ability of execution, and is restricted by syntax. In a word, BPEL has less expressive power than BPMN. We will apply and improve flatting strategy [4] and specify transformation from BPEL to BPMN.

3.1 The mapping between BPEL and BPMN

By analyzing elements in BPMN and structures in BPEL, we divide their model elements into 8 categories[5,6]. They are defined as follows:

- Flow objects
- Connection objects
- Swim lanes
- Artifacts
- Reusable information object, such as "variable" and "correlationSet" in BPEL.
- Coordination and graph information
- Supplementary information, such as version number, author's note.
- Complex type information, such as some complex variables in BPEL, whose schemas need to be defined in WSDL.

The first 4 categories refer to the basic graph elements in BPMN. The fifth ensures correct process execution. The last three categories define supplementary information.

The BPMN has standard process graph elements, and every basic element represent a kind of annotation. With the BPEL-2-BPMN mapping, we can map the element in BPEL to the corresponding annotation in BPMN. Besides, the coordination and graph size information are essential information for the visualization of BPMN. this information is automatically added during transformation.

Table 1 The mapping between BPEL and BPMN in the eight aspects

	Elements in BPEL	Elements in BPMN
1. Flow objects	<receive>,<reply>,<invoke>	Activity： Task
	<scope>	Activity： SubProcess
	<while>	Activity： Loop
		Activity: Multi instance
	<compensate>	Activity: Compensation
	<wait>,<onMessage>	(Event)
	<while>,<switch>	XOR (Data-based)
	<pick>	XOR (Event-based)
		OR
	<flow>	AND
		Complex
2. Connection objects	Activities Order	sequence flow
	<while> condition, <case>	conditional sequence flow
	<otherwise>	default sequence flow
	message	message flow
		association
3.Swim lanes	<process>	Pool
		Lane
4. Artifacts		Data object
		Group
	<documentation>	Text annotation
5. Reusable information object	Assign	
	Web Service	<partnerLink>, <portType>, <operation>
	correlationSet	correlation
	variable	
	Property	Property
	propertyAlias	
6. coordination and graph information		Shape and size of the graph
		Coordination of the graph
7 · Supplementary information		The name of the tool and version number
8.complex type information	The definition of complex message type	

3.2 Description of transformation algorithms

1)Parse the BPEL model into tree structure.

2)Do recursive traversal of all nodes from top down, and transform the node recursively into BPMN elements.

3.3 "Link" activity

3.3.1 Introduction of link activity

"Link" activity (link) is a flexible element in BPEL and only appears in flow structure. Link can be used within concurrent activities to define arbitrary control structures[1]. When the transition-Condition is meet, the target activity will be executed. If there are more than one links pointing to the same target activity, join-condition is used.

Directed diagram can handle link easily, but not in BPMN. BPMN requests that every activity's out-degree and in-degree can only be 1. Thus, we should use gateway to realize Link. In this paper, we consider possible scenarios of "link" and use OR and AND to realize the transformation.

3.3.2 The algorithm of Link transformation

When processing a node, if this node is the target of a link, we will check out whether there is a join-condition. If it's true, an OR-gateway is added before this activity, else an AND-gateway is added before this activity.

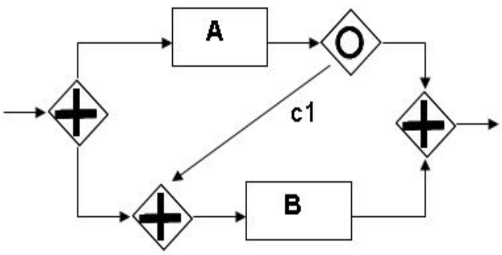
If the activity is the source of a link, whether there is a transition-condition is checked. If it's true, an OR-gateway is added after this activity, else an AND-gateway is added after this activity.

The possible transformations are shown as follows:

1). "link" set transitionCondition：

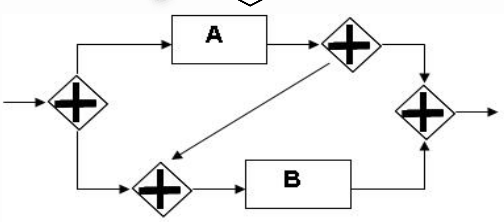
```
<flow>
  <link name="link1"/>
  <invoke name="A">
    <source    linkName="link1"
     transitionCondition="c1"/>
  </invoke>
  <invoke name="B">
    <target linkName="link1" />
  </invoke>
</flow>
```

⇩



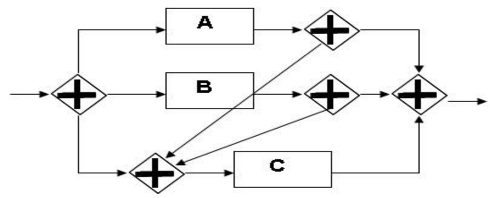
2) "link" set no transitionCondition:

```
<flow>
   <link name="link1"/>
   <invoke name="A">
     <source linkName="link1"/>
   </invoke>
   <invoke name="B">
       <target linkName="link1" />
   </invoke>
</flow>
```



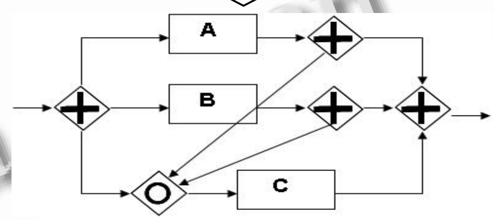
3) A target activity with more than one source and set joinCondition:

```
<flow>
   <link name="link1"/>
   <link name="link2"/>
   <invoke name="A">
     <source linkName="link1"/>
   </invoke>
   <invoke name="B">
     <source linkName="link2"/>
   </invoke>
     <invoke name="C" joinCondition
       ="bpws:getLinkStatus('link1')
       and bpws:getLinkStatus('link2')>
       <target linkName="link1" />
         <target linkName="link2" />
   </invoke>
</flow>
```



4). A target activity with more than one source，and set no joinCondition:

```
<flow>
<link name="link1"/>
<link name="link2"/>
<invoke name="A">
<source linkName="link1"/>
</invoke>
<invoke name="B">
<source linkName="link2"/>
</invoke>
<invoke name="C" joinCondition
="bpws:getLinkStatus('link1')
or bpws:getLinkStatus('link2')>
<target linkName="link1" />
<target linkName="link2" />
</invoke>
</flow>
```



3.4 Transformation algorithm

3.4.1 Some definitions

To describe our algorithm precisely, we need a set of notation and syntax. Jan Mendlings in Ref.[4] provide his definition. We improve their definition and make it more adaptable to the transformation from BPEL to BPMN.

Definition 1 (predecessor and successor nodes): N is a set of nodes, $A \subseteq N \times N$ is the arcs. The set of predeces- sor nodes pre(n)= $\{x \in N \mid (x,n) \in A\}$, and the set of successor nodes: success(n)= $\{x \in N \mid (n,x) \in A\}$。

Definition 2 (BPMN Process Graph): BPMNPG=(S, E,F,C,l,A,g). It consists of set of nodes:S, E, E, C, map- ing l: $C \rightarrow$ {AND;DXOR;EXOR;OR }, a relation:$A \in$ (S

$\bigcup$ F $\bigcup$ C)(E $\bigcup$ F $\bigcup$ C), another mapping g :

A $\rightarrow$ expr.

S: the set of start events. |S| $\geq$ 1and $\forall$ sS:pre(s)= 0 $\wedge$ successor (s)=1。

E: the set of end events. |E| $\geq$ 1and $\forall$ eE: pre(e)= 1 $\wedge$ successor(e)=0。

F: the set of middle event, including Intermediate Event and Task. $\forall$ f $\in$ F: pre(f)=1 $\wedge$ successor(f)=1。

G: the set of gateway.gG:pre(g)=1 $\wedge$ successor(g)> 1pre(g)>1successor(g) =1。

l: the type of g $\in$ G: AND, DXOR(Databased), EXOR(Eventbased), OR.

A: the set of SequenceFlow. $\forall$ n $\in$ (E $\bigcup$ F $\bigcup$ G):(n, n) $\notin$ A()(no reflexive arcs), and x, $\in$ (E $\bigcup$ F $\bigcup$ G):|{(x,y) |(x,y) $\in$ A}|=1(no multiple arcs).

c: the guard condition of a $\in$ A. expr denotes a logical expression that defines the guard condition.

Definition 3 (BPEL Control Flow): BCF=(Sequence, Flow, Switch, While, Pick, Scope, Basic, Empty, Terminate, Link, linkCond, joinCond, decomp). The set of structured activities:Struct=

Sequence $\bigcup$ Flow $\bigcup$ Switch $\bigcup$ While $\bigcup$ Pick $\bigcup$ Scope.The set of basic activities: Basisc=

Basic $\bigcup$ Empty $\bigcup$ Terminate. The set of activities: Act= Struct $\bigcup$ Basisc. Link $\subseteq$ ActxAct, linkCond: Link $\rightarrow$ expr, join Cond:A $\rightarrow$ exp, and decomp: S $\rightarrow$ P(A)\ $\varnothing$ 。

Sequence: the set of sequence activities.

Flow: the set of flow activities.

Switch: the set of switch activities.

While: the set of while activities.

Pick: the set of pick activities.

Scope: the set of scope.

Basic: the set of basic activities without terminate and empty.

Empty: the set of empty activities.

Terminate: the set of terminate activities.

Link: the set of link.

linkCond: the transition condition of link

joinCond: the join condition of link.

decomp: a mapping, from a structured activity to the set of nested activities which are the sub-activities of the structured activity.

Definition 4 (join condition): joinCond: Actexpr，as an activity x, its predecessor activities:

pre(x)={ y_1, … , y_n }. joinCond (x) = linkCond (y_1, x) $\wedge \cdots \wedge$ linkCond(y_n, x) (AND), joinCond (x) = linkCond(y_1, x) $\vee \cdots \vee$ linkCond (y_n, x) (OR)

Definition 5 (Mapping function map). The transformation function map: Basic $\rightarrow$ F.

It defines how to transform the basic activities in BPEL into BPMN diagram.

3.4.2 Algorithms

The transformation from BPEL to BPMN uses flatting strategy. It transforms BPEL Control Flow (BCF) into BPMN Process Graph.(BPG).

Algorithm 1 transforms the whole BPEL process flow. It defines the root, and transfer the function transformBCF(activity, predecessor, successor, BPG). Then do recursive traversal of all nodes from top down. The recursive traversal will be defined in algorithm 2. After BPMN graph is generated, function addCoordinate will add coordination and size for every graph. This function is essential in transformation. Only with these essential information, BPMN Process Diagram can be displayed properly.

Algorithm 1： flatten BCF

flatten(*BCF*){

Struct $\leftarrow$ *Sequence* $\bigcup$ *Flow* $\bigcup$ *Switch* $\bigcup$ *While* $\bigcup$ *Pick* $\bigcup$ *Scope*;

$S \leftarrow \{s\}$; $E \leftarrow \{e\}$; $F \leftarrow \varnothing$; $G \leftarrow \varnothing$; $A \leftarrow \varnothing$;

$root \leftarrow a$;（$a \in Struct \wedge \neg\exists s \in Struct : decomp(s) = a$）

//*a*: *BCF* structured activities

transformBCF (*root*, *s*, *e*, BPG);

for all $(l_1, l_2) \in Link$ **do**

//if link activities excise, connect the activities in BPMN

$A \leftarrow A \bigcup \{(g_1, g_2)\}$;

$c(g_1, g_2) = linkCond(l_1, l_2)$;

end for

addCoordinate(BPG);

//add coordination and size for every graph

return BPG ;

}

Algorithm 2 translates activity into BPMN process graph and connects it with predecessor and successor. As root, it will connect s and e. The last argument BPG denotes the BPMN Process Graph which has transformed and this function will base on this graph and do more transformation.

As the algorithm shows, it will check and deal with "link" first. The idea how to deal with "link" has been shown in 3.3.2. Then it transfer 5 functions to deal with 5 structures in BPEL, including sequence, switch, while, pick and flow. When the activity is Scope, it just directly handles the sub-activities that excise in scope. Basic activities will be mapped into BPMN graph directly with the function map. Besides, activities "empty" and "terminate" will be deal with as show in algorithm.

Algorithm 2： transformBCF

transformBCF(*activity*, *pred*, *succ*, BPG)
{
　if $\exists(l_1, activity) \in Links$ **then**
　　//if the *activity* is target in a link.
　　　$l(g_1) = joinCond(activity)$;
　　　if $l(g_1)$ g_1=DXOR; **else** g_1=AND; **end if**
　　　$G \leftarrow G \cup \{g_1\}$; $A \leftarrow A \cup \{(pred, g_1)\}$; $pred \leftarrow g_1$;
　end if
　if $\exists(activity, l_2) \in Links$ **then**
　　//if the activity is source in a link
　　$l(g_2) = linkCond(activity)$;
　　　if $l(g_2)$ g_2=DXOR; **else** g_2=AND; **end if**
　　　$G \leftarrow G \cup \{g_2\}$; $A \leftarrow A \cup \{(g_2, succ)\}$; $g_2 \leftarrow succ$;
　　end if
　else if $activity \in Seq$ **then**
　　　//transform Sequence activities
　　　BPG ← **transformSeq**(*activity*, *pred*, *succ*, BPG);
　　else if $activity \in Switch$ **then**
　　　//transform Switch activities
　　　BPG←
　　　　transformSwitch(*activity*, *pred*, *succ*, BPG);
　　else if $activity \in While$ **then**
　　　// transform While activities
　　　BPG ←
　　　　transformWhile(*activity*, *pred*, *succ*, BPG);
　　else if $activity \in Pick$ **then**
　　　// transform Pick activities
　　　BPG ←
　　　　transformPick(*activity*, *pred*, *succ*, BPG);
　　else if $activity \in Flow$ **then**
　　　// transform Flow activities
　　　BPG ←
　　　　transformFlow(*activity*, *pred*, *succ*, BPG);
　　else if $activity \in Scope$ **then**
　　　// transform Scope activities
　　　BPG ←
　　transformBCF(*decomp*(*activity*), *pred*, *succ*, BPG);
　　else if $activity \in Basic$ **then**
　　　//definition 5 *map.* Transform basic activities
　　　$F \leftarrow F \cup \{map(activity)\}$;
　　　$A \leftarrow A \cup \{(pred, activity), (activity, succ)\}$;
　　else if $activity \in Empty$ **then**
　　　//transform Empty
　　　$A \leftarrow A \cup \{(pred, succ)\}$;
　　else if $activity \in Terminate$ **then**
　　　//transform Terminate
　　$E \leftarrow E \cup \{e\}$; $A \leftarrow A \cup \{(pred, e)\}$;
　　end if
　return BPG
}

There are 5 functions to deal with BPEL structured activities, including: transformSeq, transformSwitch, transformWhile, transformPick, transformFlow. We will use transformPick as an example (Algorithm 3).

Algorithm 3： transform Pick

transformPick (*activity*, *pred*, *succ*, BPG)
{
　$de \leftarrow decomp(activity)$;
　$G \leftarrow G \cup \{g_1, g_2\}$;
　$l(g_1) = EXOR$;　$l(g_2) = DXOR$;
　　//add Event-based XOR before activity
　$A \leftarrow A \cup \{(pred, g_1), (g_2, succ)\}$;
　　//and add Data-based XOR after acticity
　for all *current* $\in$ *de* **do**
　　BPG ← **transformBCF**(*current*, g_1, g_2, BPG);
　end for
　return BPG ;
}

3.4.3 Transformation procedure

```
<sequence>
  <switch>
     <case>
        <sequence>
          <receive />
          <while>
              <invoke />
          </while>
        </sequence>
     </case>
     <otherwise>
     <sequence><empty
              />
     </sequence>
     </otherwise>
  </switch>
  <scope>
     <pick>
        …
     </pick>
    <reply />
  </scope>
</sequence>
```

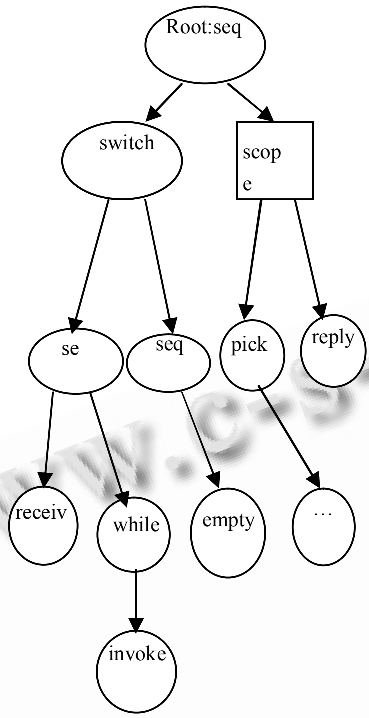


Fig.1　BPEL process flow and its tree structure

We will use an example to illustrate this procedure. For clearer description of the BPEL Control Flow, we parse it into a tree structure (Fig.1). Every node in the tree is structured activity. The whole procedure is shown in Fig.2:

1) transfer algorithm1, set s and e, and set root:seq as the first argument. Then transfer function transformBCF().

2) deal with the activity sequence, transfer function transformSeq(), and do recursive traversal of all nodes with function transformBCF().

3) deal with the activity switch, transfer function transformSwitch(), and add and-gateway.

4) deal with the activity switch.

5) Transform otherwise created in 3).

6) When scope, just transform the nodes below. Firstly transform pick, and then reply.

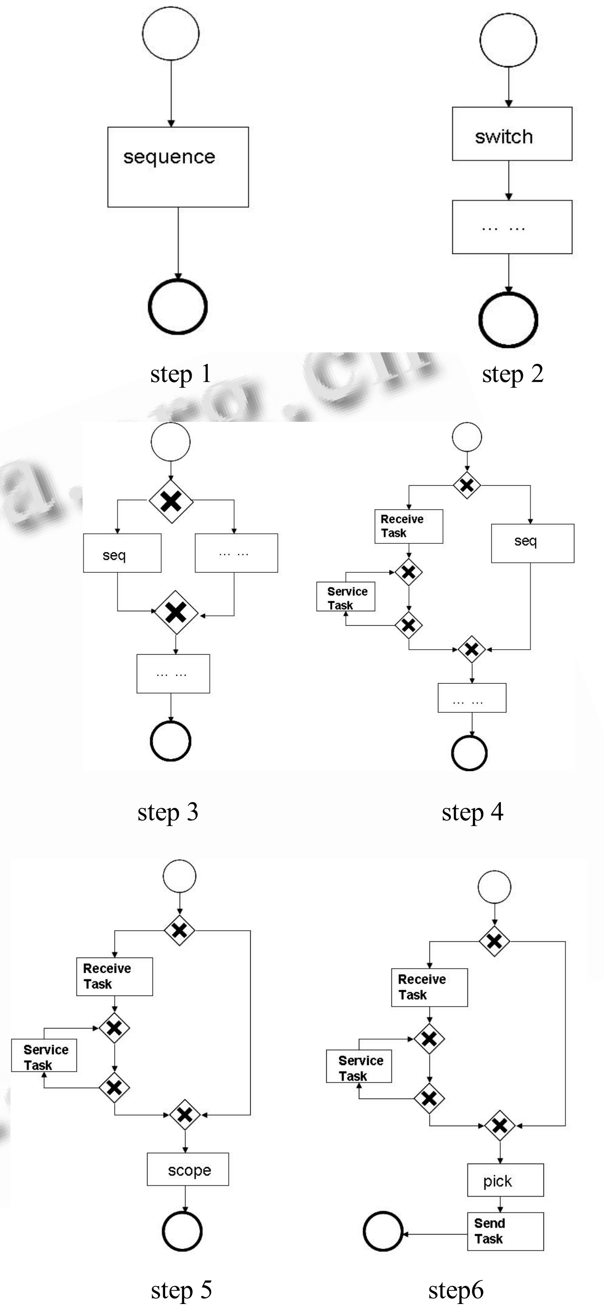


Fig.2　The algorithm procedure

4　Implementation

4.1 System architecture

The architecture is shown in Fig.3. BPEL2BPMN transformer transform BPEL Model into BPMN Model and then the BPMN model is delivered to BPEL monitor graph maker (implemented in actionscript 3.0) through network communication (tomcat). The graph maker parses the BPMN model and uses this information to generate process view which is later shown in web page. Meanwhile, we deploy the BPEL on BPEL driver and store the runtime information in database (mysql). We get this information through JDBC and deliver it to control block in the network. The control block (built by actionscript 3.0) exchange the information with monitor information block and refresh the monitor process graph.

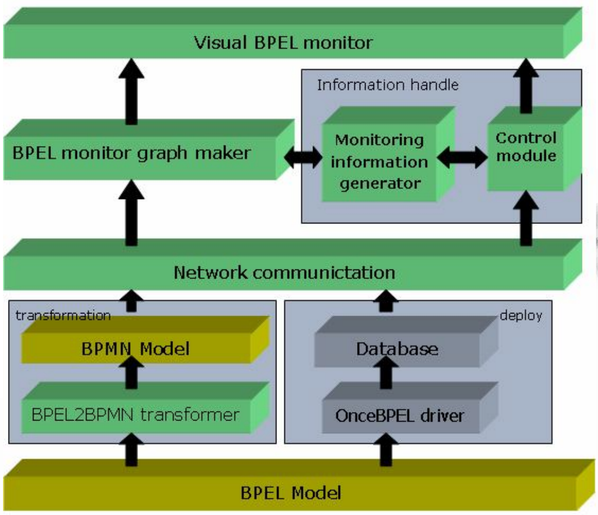
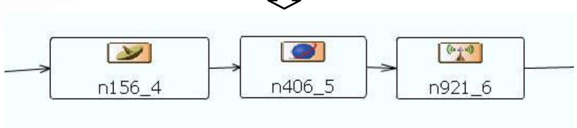


Fig.3 System architecture

4.2 Transformation Result

After completing the tool, we transform many BPEL process Into BPMN graph, and through the graph we can monitor the corresponding BPEL process visually. We present a section from a factual BPEL process as follows. It is a nested BPEL structure with scope and sequence structure activities. With the tool, we can transform it into a flat BPMN structure, and then present it in a diagram visually.

```
<scope name="SignIn">
      <sequence name="sequenceComponent_1">
        <receive name="n156_4" partnerLink="Front"
portType="tns:TravelPlanPT" operation="logon"
variable="logonRequest" createInstance="yes" />
        <assign>
          <copy>
            <from variable="logonRequest"
part="logonid" />
            <to variable="loginRequest" part="arg0" />
          </copy>
        </assign>
        <assign>
          <copy>
            <from variable="logonRequest"
part="logonpwd" />
            <to variable="loginRequest" part="arg1" />
          </copy>
        </assign>
        <invoke name="n406_5" partnerLink="PLn171_2"
portType="NSn171_2:com.once.adventure.loginservice.Logi
nServicePortType" operation="login"
inputVariable="loginRequest"
outputVariable="loginResponse" />
        <assign>
          <copy>
            <from variable="loginResponse" part="result"
/>
            <to variable="logonResponse"
part="outcome" />
          </copy>
        </assign>
        <reply name="n921_6" partnerLink="Front"
portType="tns:TravelPlanPT" operation="logon"
variable="logonResponse">
          <correlations>
            <correlation set="interaction" initiate="yes" />
          </correlations>
        </reply>
        <assign>
          <copy>
            <from expression="true()" />
            <to variable="connected" />
          </copy>
          <copy>
            <from expression="true()" />
            <to variable="isTraveBooking" />
          </copy>
        </assign>
      </sequence>
    </scope>
```



As the graph shows, the nested BPEL process section is transformed into a flat graph. When the BPEL process is executing, the tool would get he process's dynamically executive message and show that in this graph.

5 Conclusion and Future Work

In this paper, we apply a flatting algorithm and transform nested BPEL process control flow into flat BPMN process graph without hierarchy. Especially we analyze various scenarios of the transformation of link activity, provide a method to deal with it, and add the method into the transformation algorithm.

Our analysis of the mapping from BPEL process activities to BPMN process graph element show that the mapping is not one-to-one, and some supplementary information in BPMN cannot automatically obtained from BPEL process. However this supplementary information is necessary for visual presentation, such as size and coordination. We add this function in the algorithm as well.

Finally, we present the structure of our monitoring tool which is based on our transformation algorithm, and show the transformation result of link activity.

References

1 Business Process Execution Language for Web Services(version 1.1).

2 Business Process Modeling Notation Specification.

3 Recker J, Mendling J. On the Translation between BPMN and BPEL: Conceptual Mismatch between Process Modeling Languages. Proceedings of Workshops and Doctoral Consortium for the 18th International Conference on Advanced Information Systems Engineering. Namur University Press, Luxembourg, Grand-Duchy of Luxembourg, 2006.

4 Mendling JJ, Lassen KB, Zdun U. Transformation Strategies between Block-Oriented and GraphOriented Process Modelling Languages. Multikonferenz Wirtschaftsinformatik 2006. Band 2. GITO-Verlag, Berlin, Germany, 2006:297－312.

5 Florian J, Susanne L, Gregor Z. Information losses within the collaborative integration of different process models - BPML as an XML-based interchange format for BPMN business process models. Proceedings to 40th Annual Hawaii International Conference on System Sciences (HICSS'07), 2007.

6 Mendling J, Nüttgens M. Event-Driven- Process- Chain Markup Language (EPML): Anforderungen zur Definition eines XML- Schemas für ereignisgesteuerte Prozessketten. Nüttgens M, Rump F(Ed.): Proceedings of the 1st GI Workshop on Event-Driven Process Chains, Trier 2002:87－93.