BPELVT

BPEL Verification TOOL

ReDCAD Laboratory, University of Sfax

Introduction

To ensure a reliable transformation based grammars, we focus in this mapping to keep the same semantics and the same functionality of BPEL activities in Promela code. We define a set of transformation rules to map BPEL into PROMELA code. In the following, we present the transformation rules.

1. Partnerlinks

BPEL code	Steps	Transformation rules
<bpel:partnerLink name="name_Link" partnerLinkType="name_PLT" myRole="name_MR" partnerRole="name_PR"> </bpel:partnerLink>	Step 1	partsbpel: BEGIN$_$PARTLINK name partlinktyp (myrole)? (partnerRole)? END$_$PARTLINK
	Step 2	Tokens {PARTTOKEN} partsbpel: ->^(PARTTOKEN name partlinktyp (myrole)? (partnerRole)?);

	Step 3	partsbpel : ->^(PARTTOKEN name
		partlinktyp (myrole) ?
		(partnerRole) ?)
		{
		chan name_Link_IN = [0] of {mtype, type_var};
		chan name_Link_OUT = [0] of {mtype, type_var};
		};

2. Variables

BPEL code	Steps	Transformation rules
<!- - Complexe variable - - > <bpel :variable name="name_var" messageType="msgtype_var"> </bpel :variable> <!- - Simple variable - - > <bpel :variable name="name_var" type="type_var"> </bpel :variable>	Step 1	varsbpel : BEGIN_VAR name msgtypesimple END_VAR msgtypesimple : messagetype | type
	Step 2	Tokens {VARTOKEN, MSGTYPSIMPLTOKEN} varsbpel : ->^(VARTOKEN name msgtypesimple); msgtypesimple : ->^(MSGTYPSIMPLTOKEN messagetype) | ->^(MSGTYPSIMPLTOKEN type) ;
	Step 3	variable : ->^(VARDEF name msgtypesimple){ <! - - Complexe variable - -> typedef type_msgtype_var{types_vars ;} type_msgtype_var name_var ; | <! - - Simple variable - -> type name_var ; } ;

3. Basic activities

The transformation steps for every basic activity are presented as follows:

a) Invoke

BPEL code	Steps	Transformation rules
<bpel :invoke name="name_inv" partnerLink="name_Link" operation="name_op" portType="name_PT" inputVariable="name_in_var" outputVariable="name_out_var"> </bpel :invoke>	Step 1	invoke : BEGIN_INVOKE name partlink operation porttype inputvariable (outvariable) ? END_INVOKE
	Step 2	Tokens{INVOKETOKEN}; invoke : ->^(INVOKETOKEN name partlink operation porttype inputvariable (outvariable) ?) ;
	Step 3	**Invoke (Synchrone)** invoke : ->^(INVOKETOKEN name partlink operation porttype inputvariable (outvariable) ?) { canal_name_Link_OUT! name_op, name_in_var ; canal_partlink_IN ? name_op, name_out_var ;

		} ; **Invoke (Asynchrone)** invoke : ->^(INVOKETOKEN name partlink operation porttype inputvariable (outvariable) ?) { canal_name_Link_OUT ! name_op, name_in_var ; } ;

b) Receive

BPEL code	Steps	Transformation rules
<bpel :receive name="name_rec" partnerLink="name_Link" operation="name_op" portType="name_PT" Variable="name_in_var" createInstance="yes/no" </bpel :receive>	Step 1	receive : BEGIN_RECEIVE name partlink operation porttype variable createInstance END_RECEIVE
	Step 2	Tokens{RECEIVETOKEN} ; receive : ->^(RECEIVETOKEN name partlink operation porttype variable createInstance);
	Step 3	receive : ->^(RECEIVETOKEN name partlink operation porttype variable createInstance) { canal_name_Link_IN ? name_op, name_in_var ; } ;

c) Reply

BPEL code	Steps	Transformation rules
<bpel :reply name="name_rep" partnerLink="name_Link" operation="name_op" portType="name_PT" variable="name_out_var" </bpel :reply>	Step 1	reply : BEGIN_REPLY name partlink operation porttype variable END_REPLY
	Step 2	Tokens{REPLYTOKEN}; reply : ->^(REPLYTOKEN name partlink operation porttype variable) ;
	Step 3	reply : ->^(REPLYTOKEN name partlink operation porttype variable) { canal_name_Link_OUT ! name_op, name_out_var ; } ;

d) Assign

<table>
<tr><th>BPEL code</th><th>Steps</th><th>Transformation rules</th></tr>
<tr><td rowspan="3"><bpel :assign
validate="name_valid"
name="name_assign" >

A) <!- - Copy between two variables - ->

<bpel :copy>
<bpel :from> Variable_from
</bpel :from>
<bpel :to> Variable_to
</bpel :to>
</bpel :copy>

B) <!- - Assigning a value to a variable- ->

<bpel :copy>
<bpel :from> value_from
</bpel :from>
<bpel :to> Variable_to
</bpel :to>
</bpel :copy>

C) <!- - Copy between two expressions - ->

<bpel :copy>
<bpel :from> expression_from
</bpel :from>
<bpel :to> expression_to
</bpel :to>
</bpel :copy>
</bpel :assign></td><td>Step 1</td><td>assign : BEGIN_ASSIGN validate name (copy)+
END_ASSIGN
copy : BEGIN_COPY from to END_COPY
from : BEGIN_FROM variable | value | expression
END_FROM
to : BEGIN_TO variable | value | expression
END_TO</td></tr>
<tr><td>Step 2</td><td>Tokens{ASSIGNTOKEN ; COPYTOKEN;
FROMTOKEN; CONTENTFROMTOKEN;
TOTOKEN; CONTENTTOTOKEN;};

assign : ->^(ASSIGNTOKEN validate name
(copy)+) ;
copy : ->^(COPYTOKEN from to) ;
from : ->^(FROMTOKEN contentfrom) ;
contentfrom : ->^(CONTENTFROMTOKEN
variable) |->^(CONTENTFROMTOKEN
value) | ->^(CONTENTFROMTOKEN expression)
;

to : ->^(TOTOKEN contentto) ;
contentto : ->^(CONTENTTOTOKEN variable) |
->^(CONTENTFROMTOKEN value) |
->^(CONTENTTOTOKEN expression) ;</td></tr>
<tr><td>Step 3</td><td>A) variable_to = variable_from;

B) variable_to = value_from;

C) expression_to = expression_from;</td></tr>
</table>

4. Structured activities

a) If

BPEL code	Steps	Transformation rules
<bpel :if name="name_if"> <bpel :condition> Expr_Cond </bpel :condition> activities <bpel :elseif> <bpel :condition> Expr_Cond2 </bpel :condition> activities </bpel :elseif> <bpel :else> activities </bpel :else> </bpel :if>	Step 1	if : BEGIN_IF (Expr_Cond) (activities)+ (elseif Expr-Cond2 (activities)+)* (else (activities)+) ? END_IF
	Step 2	Tokens{IFTOKEN} if : ->^(IFTOKEN (Expr_Cond) (Activities)+ (elseif (Expr-Cond2) (activities)+)* (else (activities)+) ?) ;
	Step 3	if : ->^(IFTOKEN (Expr_Cond) (Activities)+ (elseif (Expr-Cond2) (activities)+)* (else (activities)+) ?) { if : : (Expr_Cond) -> activities ; : : else -> if : : (Expr-Cond2) -> activities ; : : else -> activities ; fi; fi; } ;

b) Pick

BPEL code	Steps	Transformation rules
<bpel :pick name="name_pick"> <onMessage partnerLink="name_Link" operation="name_op" portType="name_pt" variable="name_var" > Activities </bpel :onMessage> <bpel :onAlarm (for=" .. " | until-" .. ") > Scope </bpel :onAlarm> </bpel :pick>	Step 1	pick : BEGIN_PICK name (onmsg)+ (onalarm)* END_PICK onmsg : BEGIN_MSG partlink operation porttype variable (Activities)+ END_MSG onalarm : BEGIN_ALARM foralarm Scope END_ALARM
	Step 2	Tokens{PICKTOKEN} pick : ->^(PICKTOKEN name (onmsg)+ (onalarm)*) ; onmsg : ->^(ONMSGTOKEN partlink operation porttype variable (Activities)+) ; onalarm : ->^(ONALARMTOKEN foralarm Scope) ;

	Step 3	pick : ->^(PICKTOKEN name (onmsg)+ (onalarm)*) { if : : canal_partnerlink_IN ? name_op, name_var -> Activities ; : : true -> Scope ; fi; }

c) While

BPEL code	Steps	Transformation rules
<bpel :while name="name_while"> <bpel :condition> <![CDATA[Expr_cond]]> </bpel :condition> Activities </bpel :while>	Step 1	while : BEGIN_WHILE name condition (Activities)+ END_WHILE
	Step 2	Tokens{WHILETOKEN} while : ->^(WHILETOKEN name condition (Activities)+) ;
	Step 3	while : ->^(WHILETOKEN name condition (Activities)+) { do : : Expr_cond -> Activities ; : : else -> break ; od ; } ;

d) Foreach

BPEL code	Steps	Transformation rules
<bpel :forEach parallel="no/yes" counterName="Val_Count" name="name_foreach"> <bpel :startCounterValue> <![CDATA[Start_value]]> </bpel :startCounterValue> <bpel :finalCounterValue> <![CDATA[End_value]]> </bpel :finalCounterValue> Scope </bpel :forEach>	Step 1	foreach : BEGIN_FOREACH parallel countername name startcountervalue finalcountervalue Scope END_FOREACH
	Step 2	Tokens{FORTOKEN} foreach : ->^(FORTOKEN parallel countername name startcountervalue finalcountervalue Scope) ;
	Step 3	foreach : ->^(FORTOKEN parallel countername name startcountervalue finalcountervalue Scope) { int i ; i = startcountervalue ;

		do : : i<= finalcountervalue -> Scope ; i++ ; : : else -> break; od ; } ;

e) RepeatUntil

BPEL code	Steps	Transformation rules
<bpel :repeatUntil name="name_repeatuntil"> Activities <bpel :condition> <![CDATA[Expr_cond]]> </bpel :condition> </bpel :repeatUnti>	Step 1	repeatuntil : BEGIN_REPEATUNTIL name (Activities)+ condition END_REPEATUNTIL
	Step 2	Tokens{REPEATTOKEN} repeatuntil : ->^(REPEATTOKEN name (Activities)+ condition) ;
	Step 3	repeatuntil : ->^(REPEATTOKEN name (Activities)+ condition) { do : : Activities ; : : Expr_Cond -> break ; od ; } ;

f) Sequence

BPEL code	Steps	Transformation rules
<bpel :sequence name="name_seq"> Activities </bpel :sequence>	Step 1	sequence : BEGIN_SEQUENCE name (Activities)+ END_SEQUENCE
	Step 2	Tokens{SEQTOKEN} sequence : ->^(SEQTOKEN name (Activities)+) ;
	Step 3	sequence : ->^(SEQTOKEN name (Activities)+) { Activities ; } ;

g) Scope

BPEL code	Steps	Transformation rules
<bpel :scope name="name_scop"> Variables PartnerLinks Activities </bpel :scope>	Step 1	scope : BEGIN_SCOPE name (Variables)* (PartnerLinks)* (Activities)+ END_SCOPE
	Step 2	Tokens{SCOPETOKEN} scope : ->^(SCOPETOKEN name (Variables)* (PartnerLinks)* (Activities)+) ;

	Step 3	scope : ->^(SCOPETOKEN name (Variables)*
		(PartnerLinks)* (Activities)+)
		{
		//Declaration of new process
		proctype process()
		{
		Variables ;
		PartnerLinks ;
		Activities ;
		}
		//Processus execution
		run process() ;
		} ;

h) Flow

BPEL code	Steps	Transformation rules
<bpel :flow name="name_flow"> Activities </bpel :flow>	Step 1	flow : BEGIN_FLOW name (Activities)+ END_FLOW
	Step 2	Tokens{FLOWTOKEN} flow : ->^(FLOWTOKEN name (Activities)+) ;

	Step 3	flow : ->^(FLOWTOKEN name (Activities)+)
		{
		bool process_termine = false ;
		//Declaration of new process
		proctype process()
		{
		Activities ;
		process_termine = true ;
		}
		run process() ;
		// Synchronisation
		process_termine ;
		};