



Formal Methods For Web Service Process Modeling

**Dr. Yuhong Yan
NRC-IIT**



Web Service Process Description Languages

- **Orchestration language: BPEL4WS**
- **Choreograph languages: WS-CDL, WSCI**
- **Formal models: Process Algebras, Petri nets, Automata**



Modeling and Reasoning about Web Service Processes

- **How to model the control flow and data flow**
- **What are the reasoning tasks**
- **What are the techniques for these tasks**



Web Service Process vs. Workflow Management

- **Workflow management**
 - Hybrid system of human and software applications
 - Collaborative working environment
 - Workitems-actors
 - Process descriptions similar to orchestration languages
- **Web services**
 - Automated services
 - Interoperation and service oriented architecture
 - request-response
 - Web service process engine can be similar to workflow management engine (cf. Karagiannis 2006)



Orchestration language: BPEL4WS



How BPEL looks like?

```
<process name="echoString"
  targetNamespace="urn:echo:echoService"
  xmlns:tns="urn:echo:echoService"
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">

  <partnerLinks>
    <partnerLink name="caller"
      partnerLinkType="tns:echoPLT"
      myRole="service"/>
  </partnerLinks>

  <variables>
    <variable name="request" messageType="tns:StringMessageType"/>
  </variables>

  <sequence name="EchoSequence">
    <receive partnerLink="caller" portType="tns:echoPT"
      operation="echo" variable="request"
      createInstance="yes" name="EchoReceive"/>
    <reply partnerLink="caller" portType="tns:echoPT"
      operation="echo" variable="request" name="EchoReply"/>
  </sequence>

</process>
```



Business Process Execution Language for Web Service (BPEL4WS)

The information described by BPEL4WS:

- The **execution order** of the **activities**
- The **triggering conditions** of the **activities**
- The **partners** for the external activities
- The **composition** of Web Services
- The **binding** to WSDL (activities <-> operations)

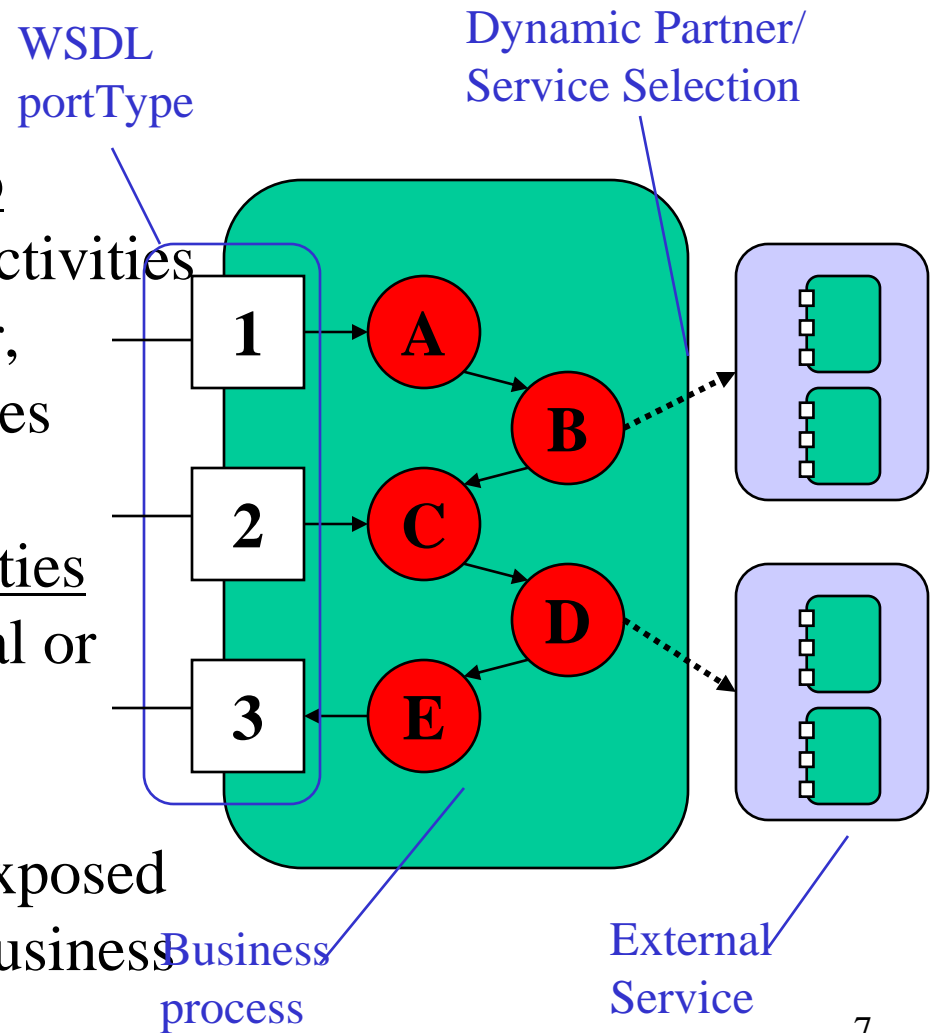
Business Process (what) versus WSDL (how)

Business Process: **what** to do

- Modeled as a sequence of activities
- Tools aid to define, monitor, and manage business processes

WSDL: **how** to execute activities

- An activity can be an internal or external Web service (SOAP/WSDL)
- A business process can be exposed for consumption by another business process or a client app





BPEL activities

- **Basic Activities: atomic actions**
 - **<receive>**
 - **<reply>**
 - **<invoke>**
 - **<assign>**
 - **<throw>**
 - **<terminate>**
 - **<wait>**
 - **<empty>**



BPEL Structured Activities

- **<sequence>**: an ordered sequence of activities
- **<flow>**: parallel activities
- **<switch>**: “case-statement” approach
- **<while>**: loop
- **<pick>**: execute one of several alternative paths



<faultHandlers>

- In response to faults
- Defines the recovery actions when faults occur
- To undo the partial and unsuccessful work of a scope in which a fault has occurred
- To do other business logic

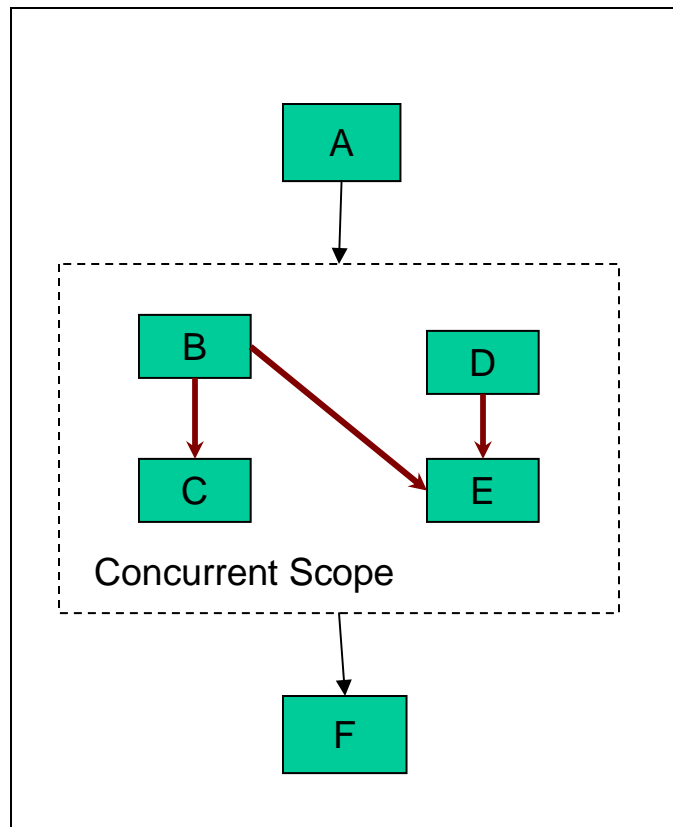


Data flow: <variable>

- **Data variables used by activities**
 - **<variable messagetype=“...”>**: WSDL message;
 - **<variable type=“...”>**: XML Schema simple type;
 - **<variable element=“...”>**: XML Schema element.
- **Variables associated with message types can be specified as input or output variables for **invoke**, **receive** and **reply** activities.**

A Business Process and BPEL Description

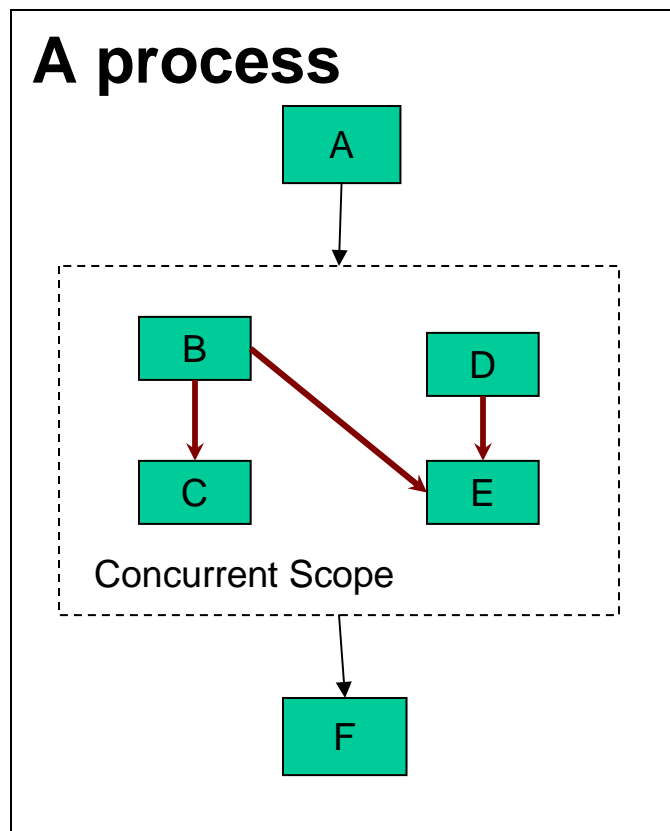
A process



```
<sequence >
  <receive a v ><corr. >p <\corr. ><\receive >
  <flow >
    <invoke b b_in b_out >
      <source linkname=b_to_c >
      <source linkname= b_to_e >
    <\invoke >
    <invoke c c_in c_out >
      <target linkname=b_to_c >
    <\invoke >
    <invoke d d_in d_out >
      <source linkname=d_to_e >
    <\invoke >
    <invoke e e_in e_out >
      <target linkname=b_to_e >
      <target linkname= d_to_e >
    <\invoke >
  <\flow >
  <reply f w><corr. >p <\corr. ><\reply >
<\sequence >
```

Control the execution order using <link>

<link> defines dependencies between the activities in BPEL



Execution orders:

A-B-C-D-E-F

A-B-D-C-E-F

A-D-B-C-E-F

A-D-B-E-C-F



Choreography languages



Web Service Choreography Interface (WSCI)

- **Describes the flow of messages**
- **The observable behaviors of a Web service**
- **No internal behaviors**
- **Compared with WSDL**
 - **WSDL does not describe the execution orders of operations**
 - **WSDL does not describe the correlation aspects**

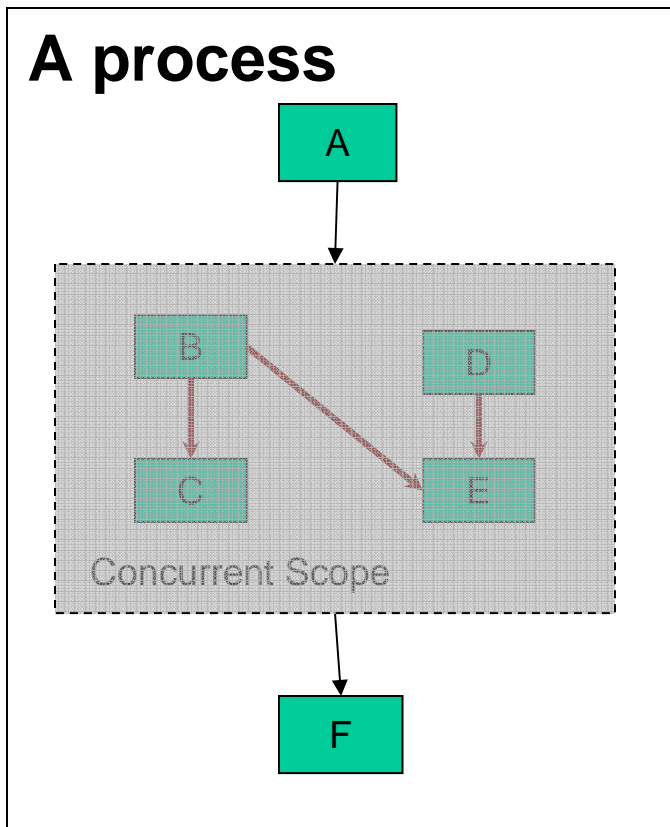


Web Service Choreography Interface (WSCI)

- Atomic Activities
 - **<action>** :mapping to an operation in WSDL
 - **<delay>**
 - **<empty>**
 - **<fault>**
 - **<call>**
 - **<spawn>**
 - **<join>**
 - Complex Activities
 - **<process>**
 - **<all>**: parallel
 - **<choice>**
 - **<foreach>**
 - **<sequence>**
 - **<switch>**
 - **<until>**
 - **<while>**
 - Variables: map to SOAP messages similar like BPEL
- } For the instances of a
<process>

A Business Process and WSCI Description

A process



<process >

<sequence >

<action a v ><corr. >p <\corr. ><\action >

<action f w ><corr. >p <\corr. ><\action >

<\sequence >

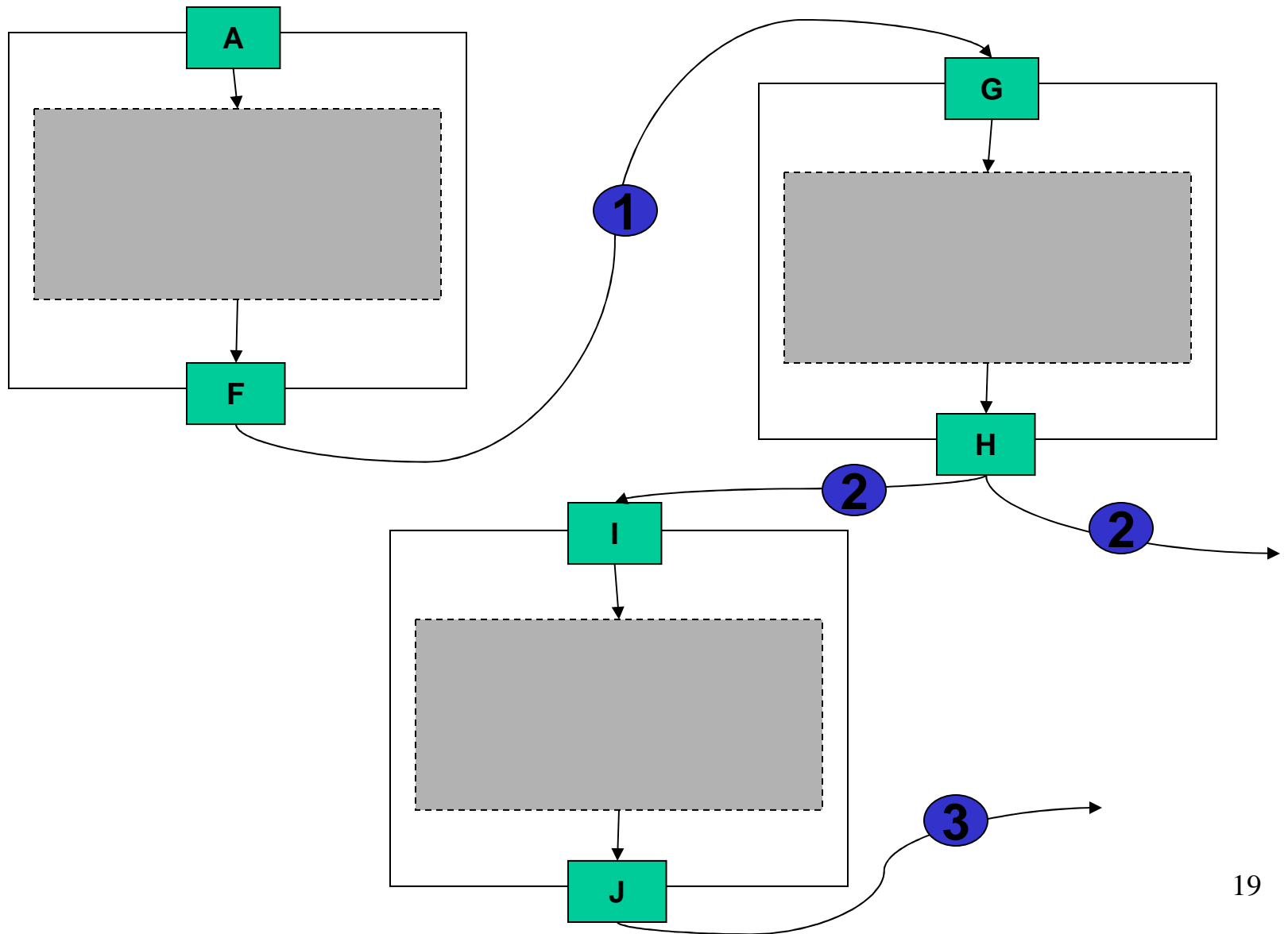
<\process >



Web Service Choreography Description Language (WS-CDL)

- **A global conversation among services**
 - **WSCI is a projection of the entire conversation on a participant**
- **Observable behaviors**
- **Based on Pi-Calculus**
- **Elements**
 - **Channels**
 - **Three structures: sequence, parallel, and choice**
 - **Variables**

A WS-CDL Example





ebXML Business Process Specification Schema (ebXML BPSS)

- **A choreography language**
- **Between two business partners**
- **Contains more information than Web service process description languages**
 - **Business documents**
 - **Business transactions
(protocol to exchange documents)**
 - **Binary collaborations
(composition of transactions)**
 - **Multiparty collaborations
(2 or more binary collaborations)**
 - **Substitution sets**



Modeling Web Service Processes with Formal Models



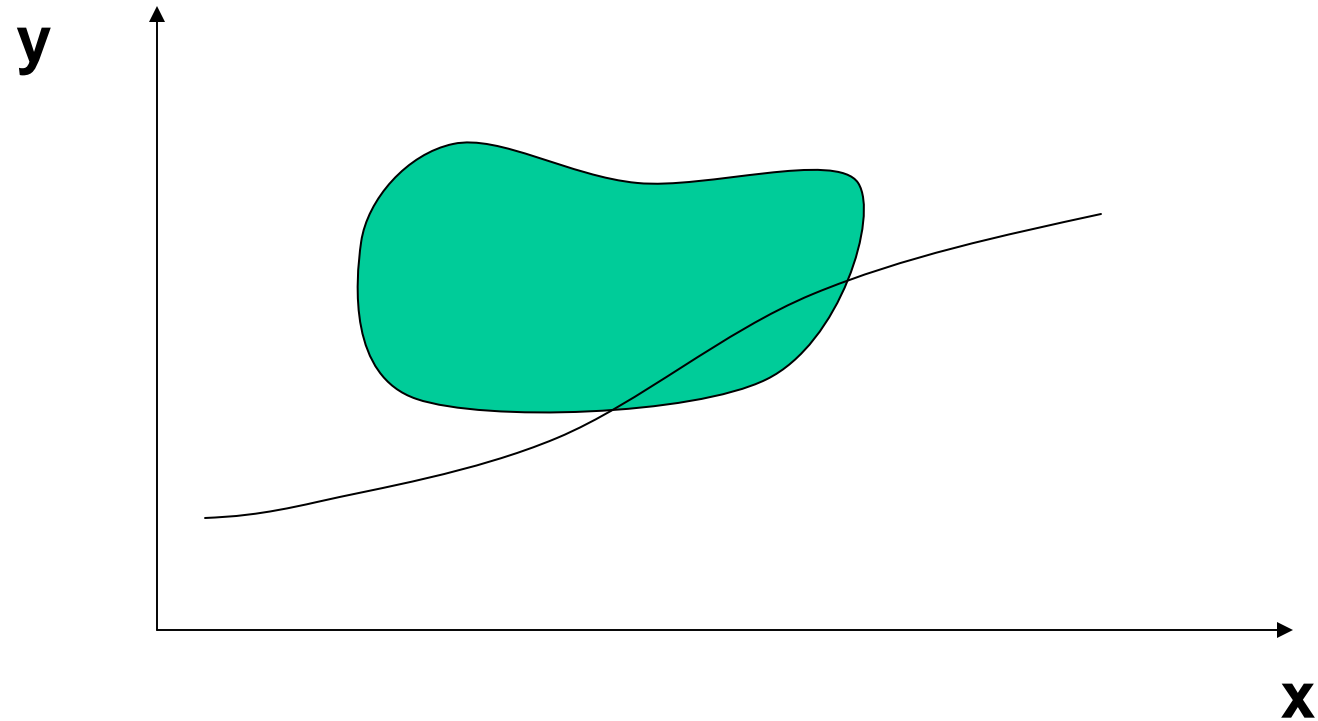
Cartesian Product

If A and B are two sets, the Cartesian product or cross product of A and B , written $A \times B$, is the set of all pairs wherein the first element is a member of A and the second element of B .

Example: if $A = \{1, 2\}$, $B = \{x, y, z\}$

$A \times B = \{(1, x), (1, y), (1, z), (2, x), (2, y), (2, z)\}$.

Function and Relation



Function:

$$f(a)=b$$

$$f:X\rightarrow Y$$

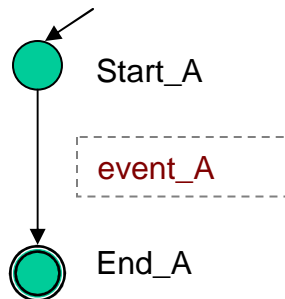
Relation:

$$p:X\times Y\rightarrow\{\text{true},\text{false}\}$$

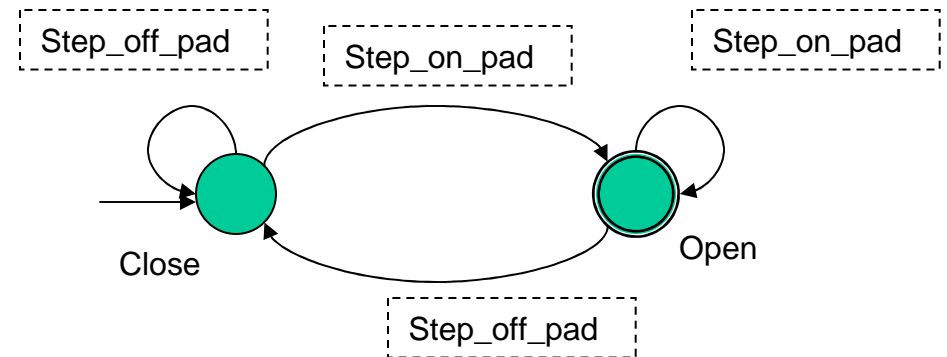
Automata

A finite automaton Γ is a 5-tuple $\Gamma = (X, \Sigma, T, x_0, F)$, where:

- ★ X is a finite set of states;
- ★ Σ is a finite set of events;
- ★ $T \subseteq X \times \Sigma \rightarrow X$ is a finite set of transitions;
- ★ $x_0 \in X$ is a initial state;
- ★ $F \subseteq X$ is a finite set of final states.



A Web Service Activity

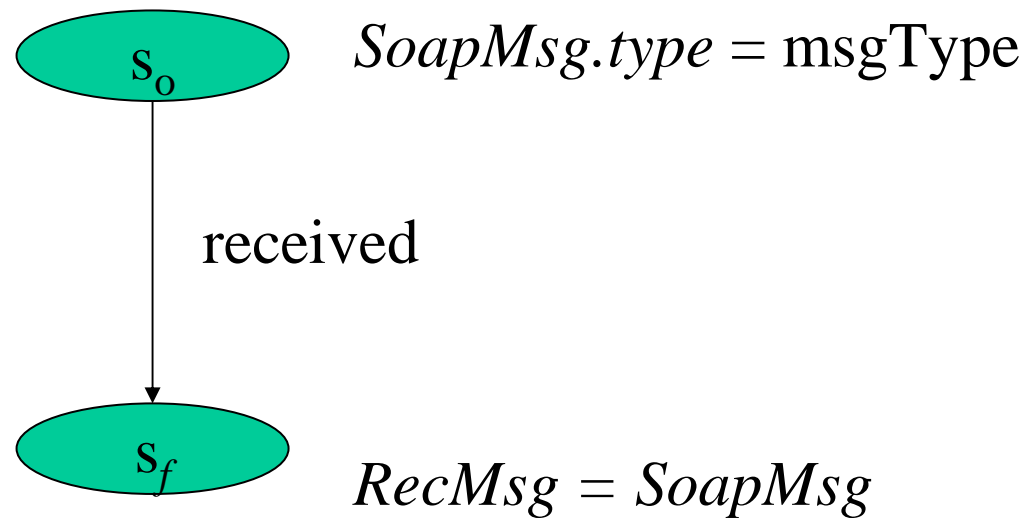


An Automatic Door

Activity

$\langle \text{receive} \rangle$

$\langle \text{receive} \rangle : \langle \{s_o, s_f\}, \{\text{received}\}, \{t\}, \{s_o\}, \{s_f\}, \mathcal{C} \rangle$
 $t : (s_o \wedge \text{SoapMsg.type} = \text{MsgType}) \xrightarrow{\text{received}} (s_f$
 $\wedge \text{RecMsg} = \text{SoapMsg})$



msgType is a predefined message type

Modeling a BPEL Activity

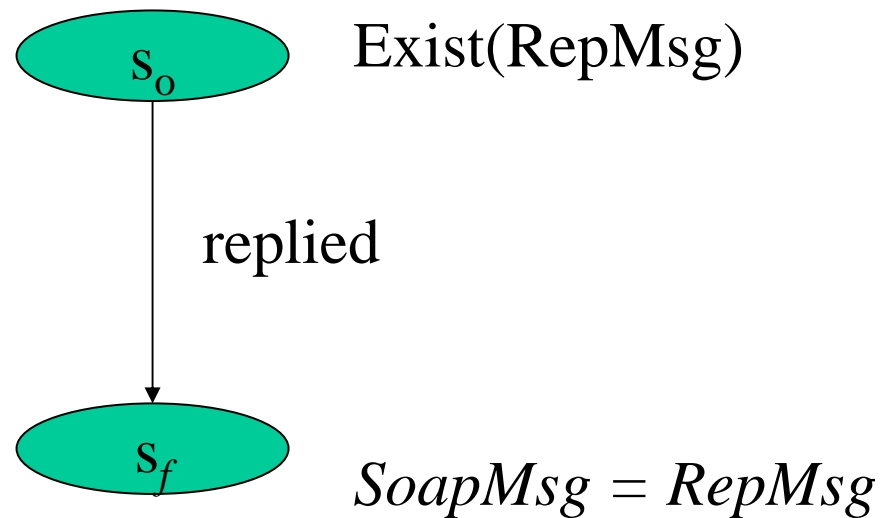
- **V**: a finite set of variables;
- **D_v**: the finite domain for the *v* in *V*;
- **C**: a finite set of constraints
 - A constraint *c* of some arity *k* is a subset of cartesian product over variables $\{v_{i1}, \dots, v_{ik}\} \subseteq V$, i.e.
$$c_j \subseteq D_{j1} \times \dots \times D_{jk}$$
 - Or a first order formula over $\{v_{i1}, \dots, v_{ik}\}$
- A transition *t*:
$$(s_i \wedge pre(V_1)) \xrightarrow{event} (s_j \wedge post(V_2))$$
- A BPEL Activity **(X, Σ, T, I, F, C)**, where
$$T \subseteq X \times \Sigma \times 2^C \rightarrow X \times 2^C$$

Activity

$\langle \text{reply} \rangle$

$\langle \text{reply} \rangle : \langle \{s_o, s_f\}, \{\text{replied}\}, \{t\}, \{s_o\}, \{s_f\}, \mathcal{C} \rangle$

$t : (s_o \wedge \text{exists}(\text{RepMsg})) \xrightarrow{\text{replied}} (s_f$
 $\wedge \text{SoapMsg} = \text{RepMsg})$



Activity

$\langle \text{invoke} \rangle$

- **Synchronous**

$\langle \{s_o, \text{wait}, s_f\}, \{\text{invoked}, \text{received}\}, \{t_1, t_2\}, \{s_o\}, \{s_f\}, \mathcal{C} \rangle$

$t_1 : (s_o \wedge \text{exists}(InVar)) \xrightarrow{\text{invoked}} (\text{wait}), \text{ and}$

$t_2 : (\text{wait}) \xrightarrow{\text{received}} (s_f \wedge \text{exists}(OutVar))$

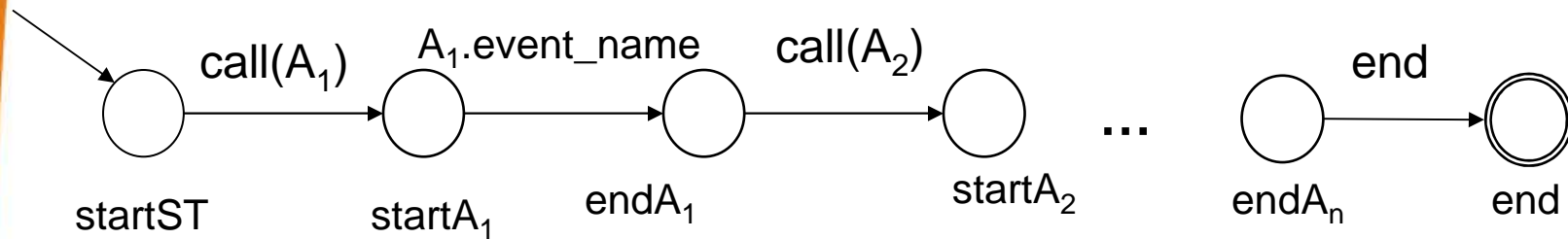
- **Asynchronous**

$\langle \{s_o, s_f\}, \{\text{invoked}\}, \{t\}, \{s_o\}, \{s_f\}, \mathcal{C} \rangle$

$t : (s_o \wedge \text{exists}(InVar)) \xrightarrow{\text{invoked}} (s_f)$

Activity

⟨sequence⟩



$\langle \{s_o, s_f\} \cup \cup S_{A_i}, \{end\} \cup \cup \{callA_i\} \cup \cup \Sigma_{A_i}, \{t_i\} \cup \cup T_{A_i}, \{s_o\}, \{s_f\}, \cup C_{A_i} \rangle$ with

$$t_0 : (s_o) \xrightarrow{callA_1} (s_{A_{1o}})$$

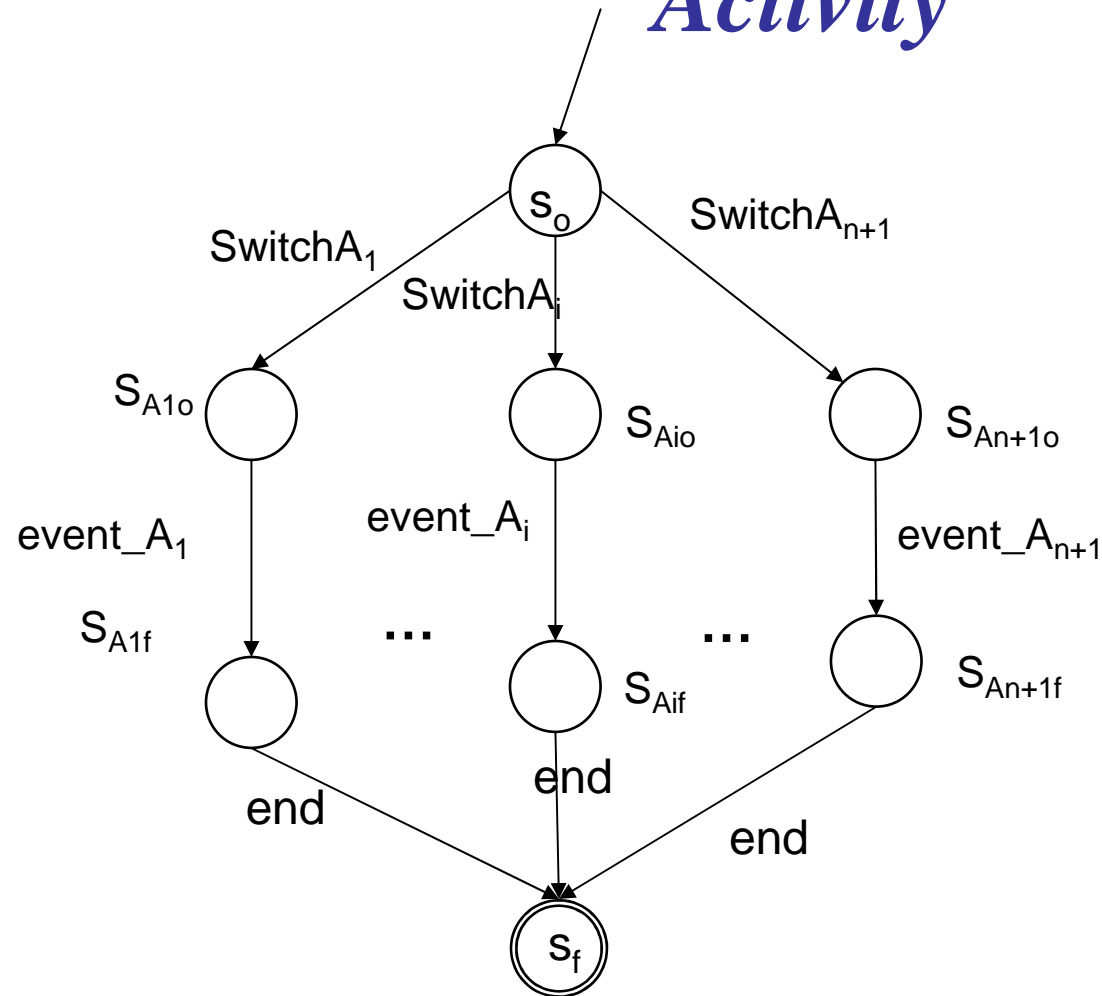
$$t_i : (s_{A_{if}}) \xrightarrow{callA_{i+1}} (s_{A_{i+1o}})$$

$$t_n : (s_{A_{nf}}) \xrightarrow{end} (s_f)$$



Activity

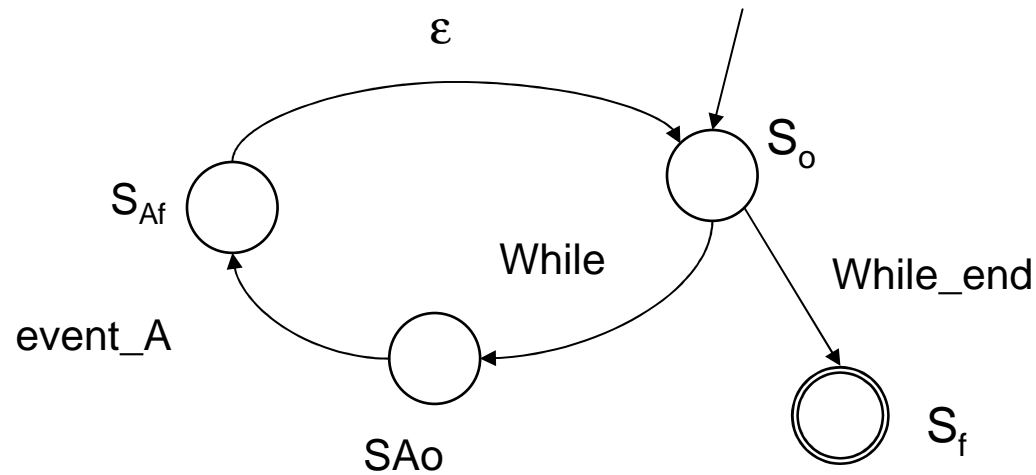
$\langle \text{switch} \rangle$



$\langle \text{switch} \rangle : \langle \{s_o, s_f\} \cup \cup S_{A_i}, \{end\} \cup \cup \{switchA_i\} \cup \cup \Sigma_{A_i}, \cup \{t_{io}\} \cup \cup \{t_{if}\} \cup \cup T_{A_i}, \{s_o\}, \{s_f\}, \cup C_{A_i} \cup \cup pre(V_i) \rangle$

Activity

⟨while⟩



while: $\langle \{s_o, s_f\} \cup S_A, \{while, while_end\} \cup \Sigma_A, \{t_o, t_f, t\} \cup T_A, \{s_o\}, \{s_f\}, \mathcal{C} \cup pre(W) \rangle$

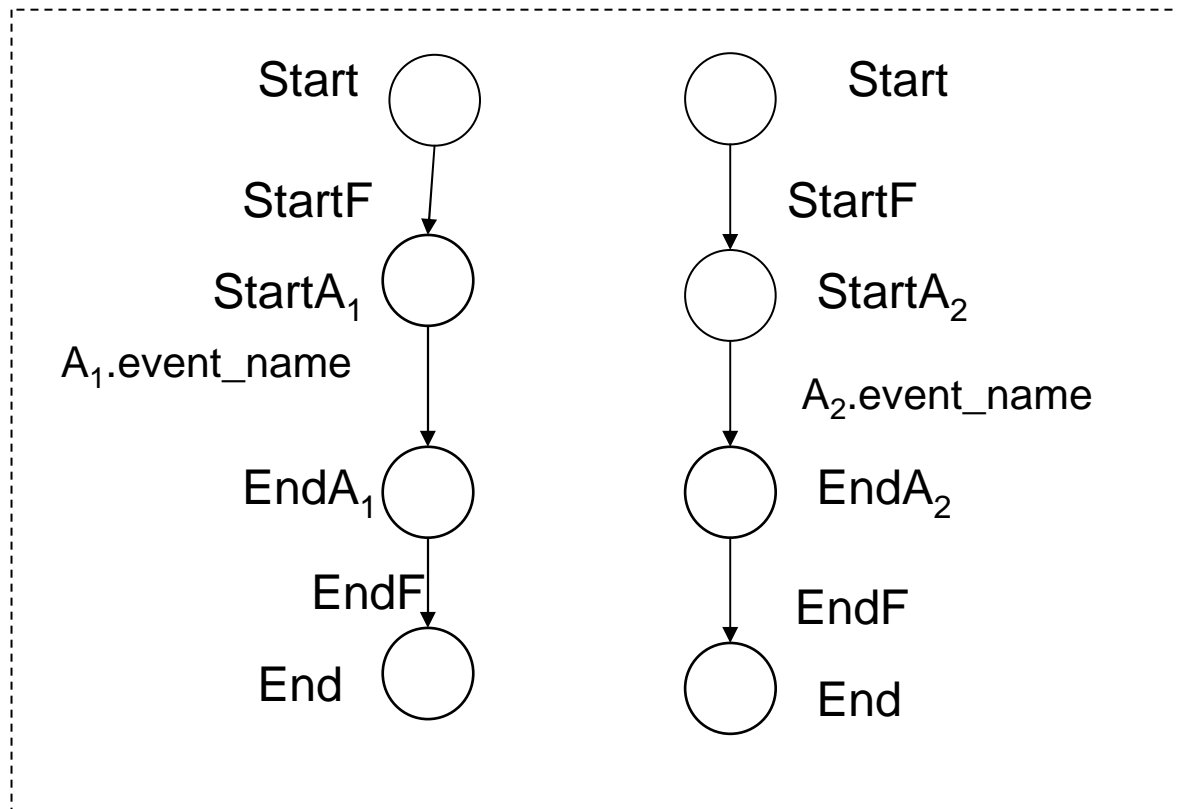
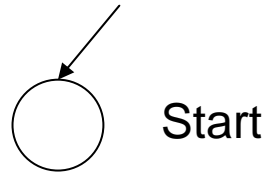
$$t_o : (s_o \wedge pre(W)) \xrightarrow{while} (s_{A_o})$$

$$t_f : (s_o \wedge \neg pre(W)) \xrightarrow{while_end} (s_f)$$

$$t : (s_{A_f}) \xrightarrow{\epsilon} (s_o)$$

Activity

⟨flow⟩

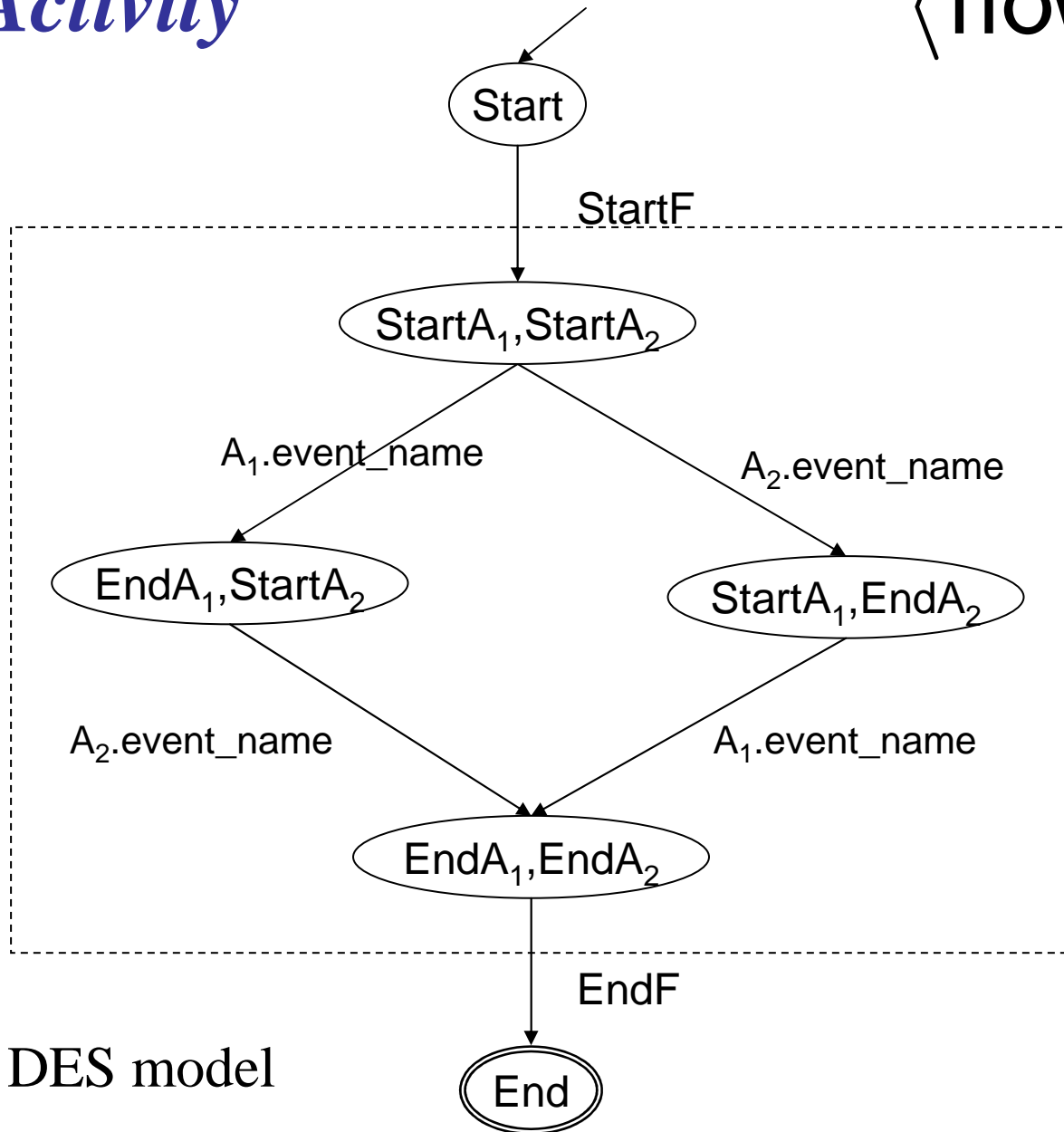


Concurrency branches



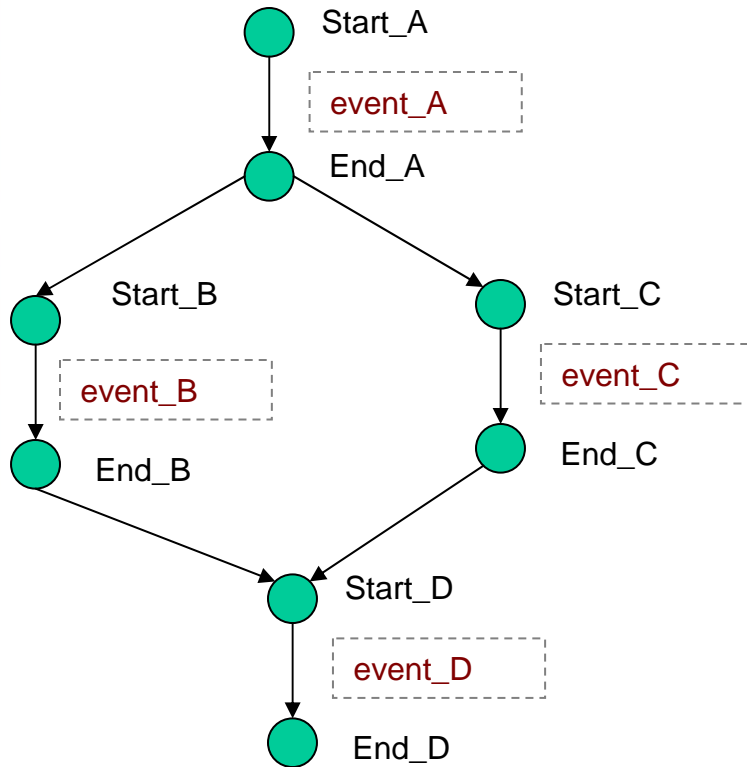
Activity

⟨flow⟩



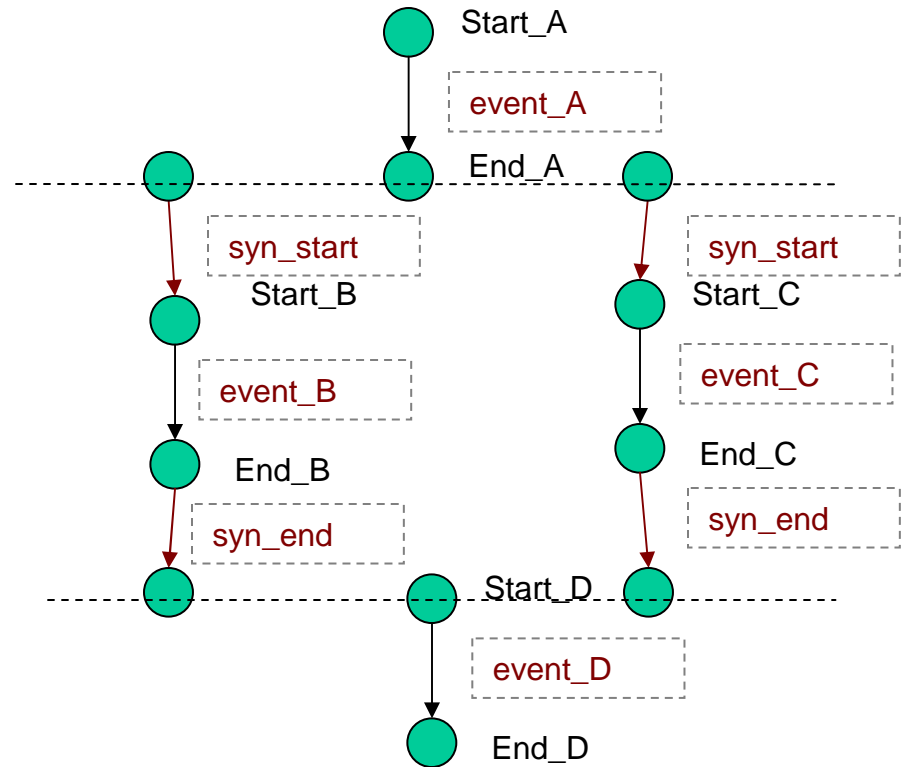
The joint DES model

Automata



Choose between B and C:

event_A - event_B - event_D or
 event_A - event_C - event_D



Parallel B and C:

event_A - syn_start - event_B - event_C - syn_end - event_D or
 event_A - syn_start - event_C - event_B - syn_end - event_D

The Automata for the Example Process

