Metode şi Algoritmi de Planificare (MAP)

2009-2010

Curs 9

Workflow Scheduling and Execution

BPEL

Agenda

- Introduction (Scheduling problem, Environment)
- DAG example
- Task Dependencies Model and DAG Scheduling
- Algorithms for DAG Scheduling
- DIOGENES DAG Framework
- Experiments
- BPEL
  - ActiveBPEL

Introduction

- Scheduling Problem
  - Allocate a group of different *tasks* on *resources*
  - NP-Complete (algorithm)
- *Task*
  - Complex description
  - Variable resource consumption
  - Dependencies: can only be started after other tasks have finished
- *Scheduling Environment*
  - Computational GRID
  - Heterogeneous - resources with different processing capacities
  - Public - resources can be used by multiple user

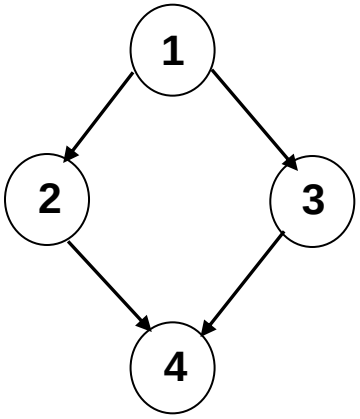
DAG example

1: a = 2

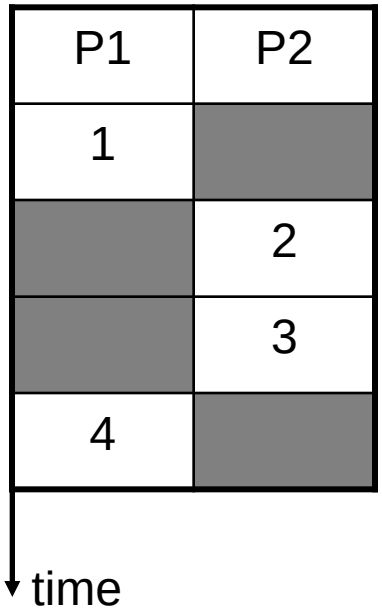
2: u = a + 2

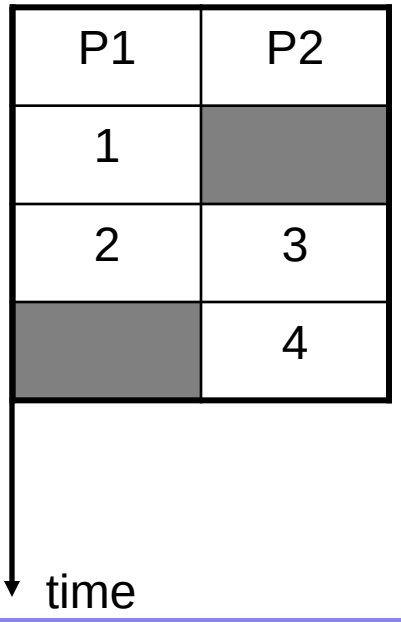
3: v = a * 7

4: x = u + v



Scheduling examples:

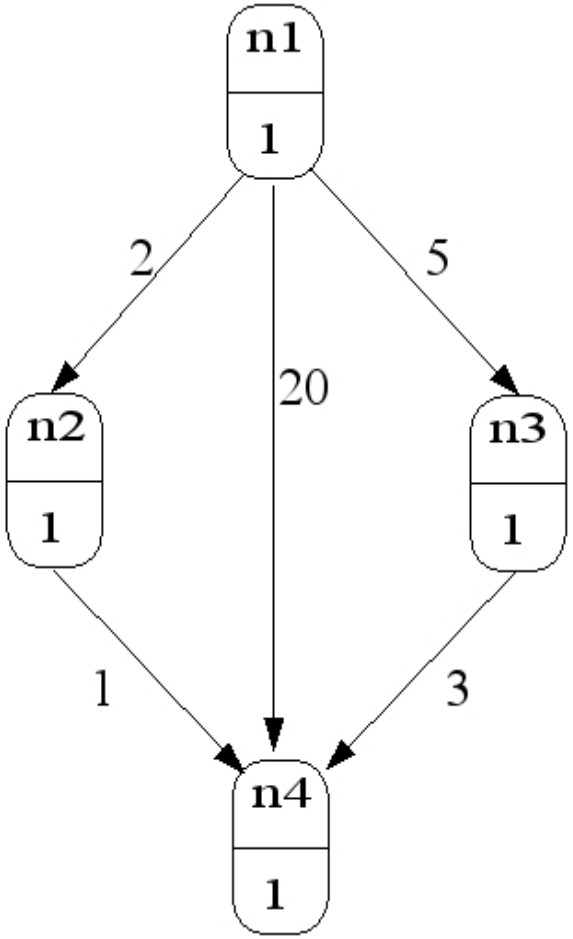




Task Dependencies Model and DAG Scheduling

- The **ALAP** (As Late As Possible) start-time of a node is a measure of how far the node's start time can be delayed without increasing the schedule length.
- The **b-level** of a node is the length of the longest path from it to an exit node.
- The **CP** (Critical Path) is the weight of the longest path in the DAG and offers an lower bound for the scheduling cost.

Task Dependencies Model and DAG Scheduling



$$ALAP(u) = CP - blevel(u)$$

Node	blevel	ALAP
1	22	0
2	3	19
3	5	17
4	1	21

DAG Scheduling Algorithms

The MCP algorithm

```
Step 1. Compute the ALAP time of each node.
Step 2.
        2.1 For each node, create a list which consists of the
            ALAP times of the node itself and all its children
            in a descending order.
        2.2 Sort these lists in an ascending lexicographical
            order.
        2.3 Create a node list according to this order.
Step 3.
       Repeat
        3.1 Schedule the first node in the node list to a
            processor that allows the earliest execution, using
            the insertion approach.
        3.2 Remove the node from the node list
       Until the node list is empty.
```

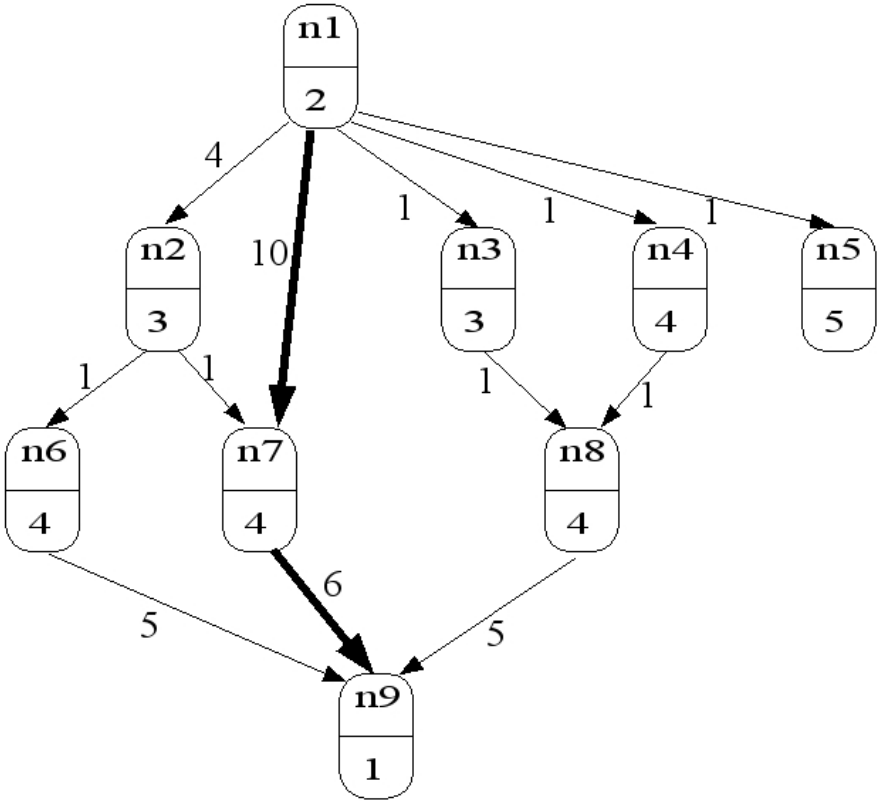
DAG Scheduling Algorithms

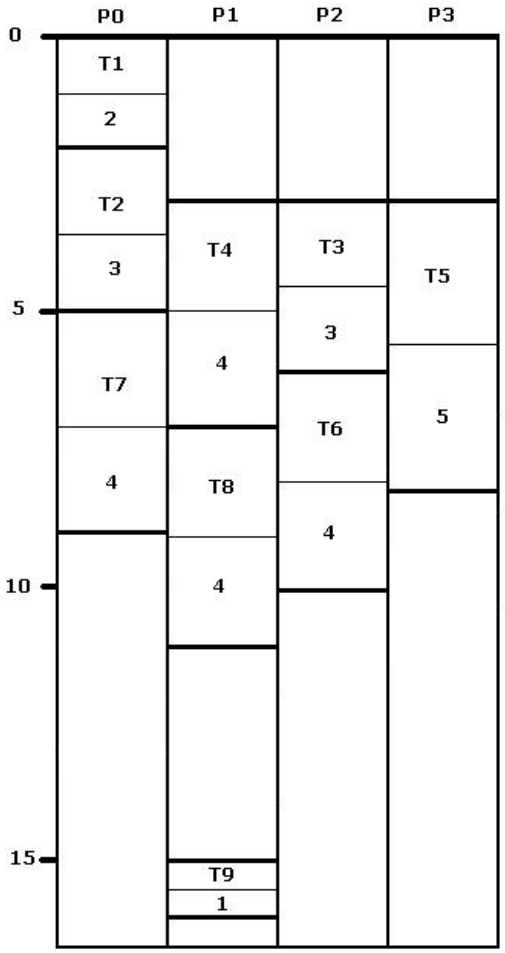
ISH(Insertion Scheduling Heuristic)	CCF(Cluster ready Children First)
•Keeps a list of tasks that are ready to be scheduled •Tries to schedule the tasks in this list, attempting to fill the empty spaces left by the previous tasks	•List based heuristic •Dynamic algorithm, able to run in parallel with the task execution.

- The result of the schedule process can be represented using a Gantt chart

DAG scheduling algorithms

- Example of a scheduled DAG using the ISH algorithm

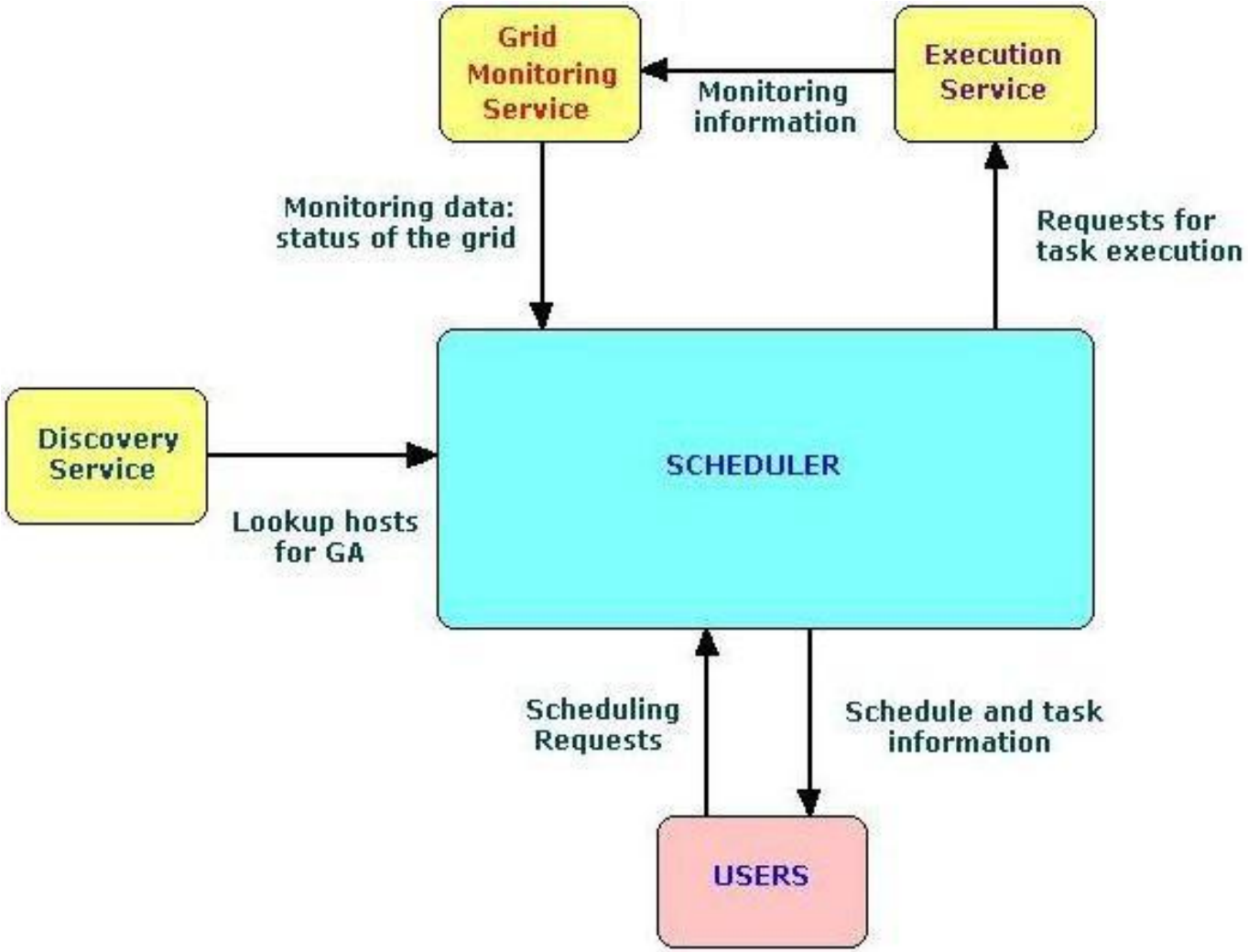




DIOGENES DAG Framework

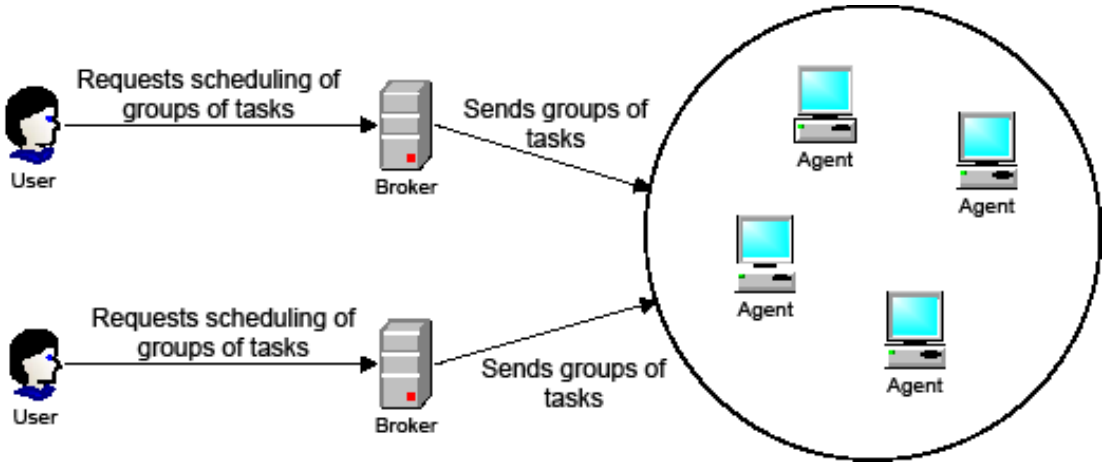
- **Discovery Service**. This service detects the available hosts where it can run the scheduling algorithm. The Discovery Service is based on JINI.
- **Grid Monitoring Service**. Returns real-time information about the various site facilities, networks, and about state of the current activities performed in the system, including the data about the scheduled tasks that were sent for execution.
- **Execution Service**. Receives the mappings returned by the scheduling algorithm and sends the tasks to run on their respective hosts.
- **Scheduler**
  - Can run a genetic algorithm for independent tasks
  - For dependent tasks it can be configured to run a DAG Algorithm

DIOGENES DAG Framework



Communication Model

- A broker receives the scheduling requests from a client and sends the tasks to be schedule to the agents
- The agents are responsible for running a scheduling algorithm for the jobs received from a broker using the monitoring information.
- A broker can run on a different remote machine or on the same machine as an agent.



Input Data Model

```
<task>
    <taskId>3</taskId>
    <path>/home/student/app/loop100.sh</p
    ath>
    <arrivingDate>2007/01/05</arrivingDat
    e>
    <arrivingTime>01:20:00</arrivingTime>
    <arguments></arguments>
    <parent>
                <Id>1</Id>
          <Cost>5</Cost>
    </parent>
    <child>
                <Id>5</Id>
          <Cost>10</Cost>
    </child>
    <requirements>
           <memory>0.0MB</memory>
          <cpuPower>2745.9404MHZ</cpuPower
    >
          <processingTime>1</processingTim
    e>
          <deadlineTime>
                              2007/01/06
    22:20:30
                </deadlineTime>
          <schedulePriority>
                -1
           </schedulePriority>
    </requirements>
    <nrexec>1</nrexec>
</task>
```

Task description

```
<Node>
<Id>1</Id>
    <FarmName>


    DIOGENESFarm
    </FarmName>
    <ClusterName>
          DIOGENESCluster
    </ClusterName>
    <NodeName>P01</NodeName>
    <Parameters>

    <CPUPower>2730.8MHZ</CPUPower>


          <Memory>512MB</Memory>
        <CPU_idle>93.7</CPU_idle>
    </Parameters>
</Node>
```

Resource description

Performance Indicators

- *Total time for the scheduling process:* is vital for applications with high priority which need scheduling right away, or applications that need only a fair estimation of the total schedule length as soon as possible.
- *Total schedule length* (SL): the time span between the start time of the first scheduled node (the root node in the DAG) and the finish time of the last leaf node in the DAG, considering that all the resources are synchronized on the same timeline
- *Normalized schedule length* (NSL): offers a good estimation of the performance of an algorithm since the Critical Path gives a lower bound to the schedule length.

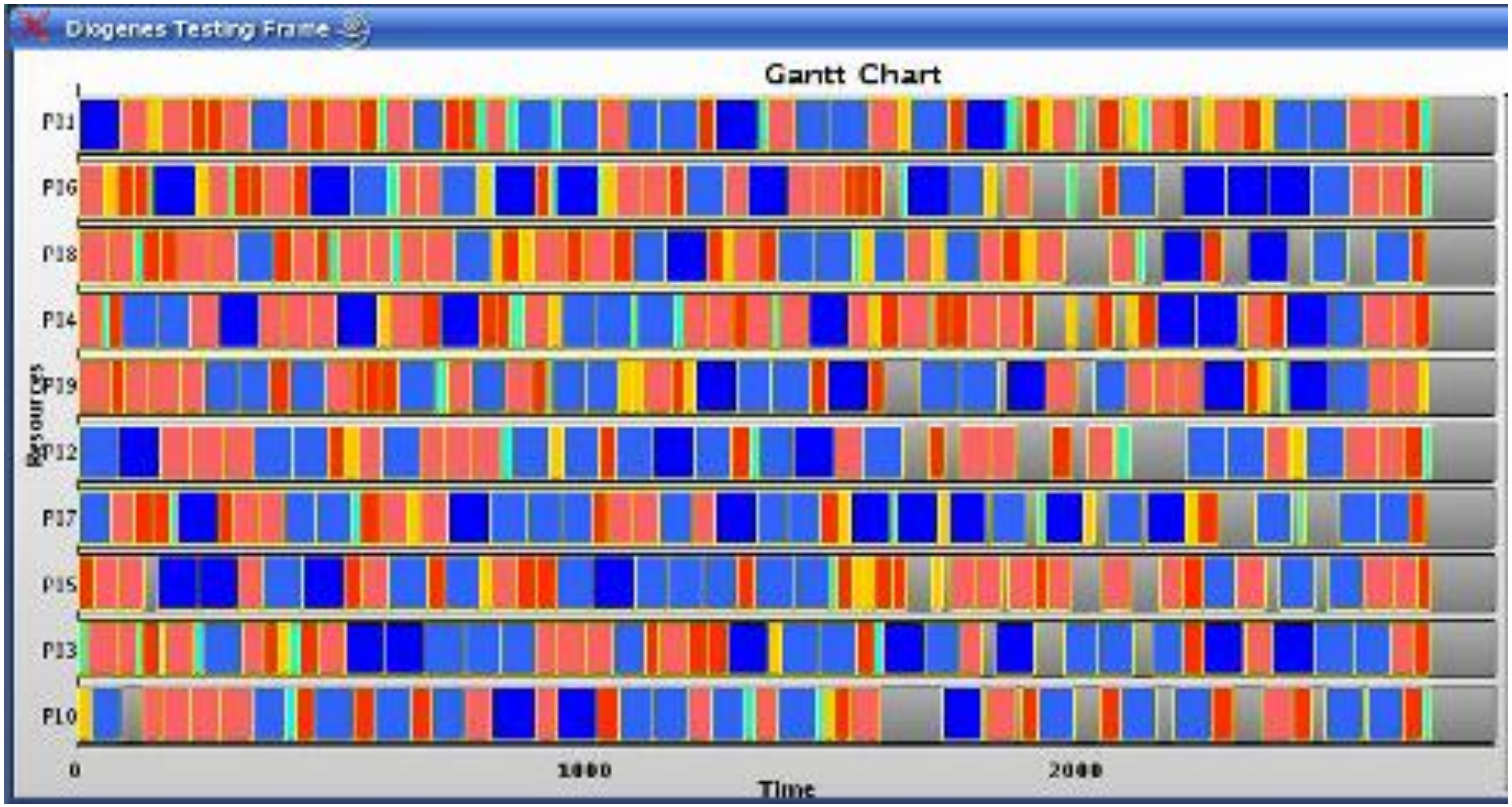
$$NSL = \frac{SL}{\sum_{n_i \in CP} w(n_i)} = \frac{SL}{CP}$$

Experimental Results

- Tasks ranging: between 100 and 1000 tasks.
- The threshold value was established to be over 0.75 for a small number of tasks, and increasing to over 0.94 for a larger number of tasks.
- Experimental tests
  - Analysis of the total time
  - Analysis of the total schedule length
  - Evaluation of the threshold variation
  - Normalized schedule length
  - Load balancing and resource allocation efficiency

Experimental Results

- Gantt chart representing the schedule for a group of dependant tasks.

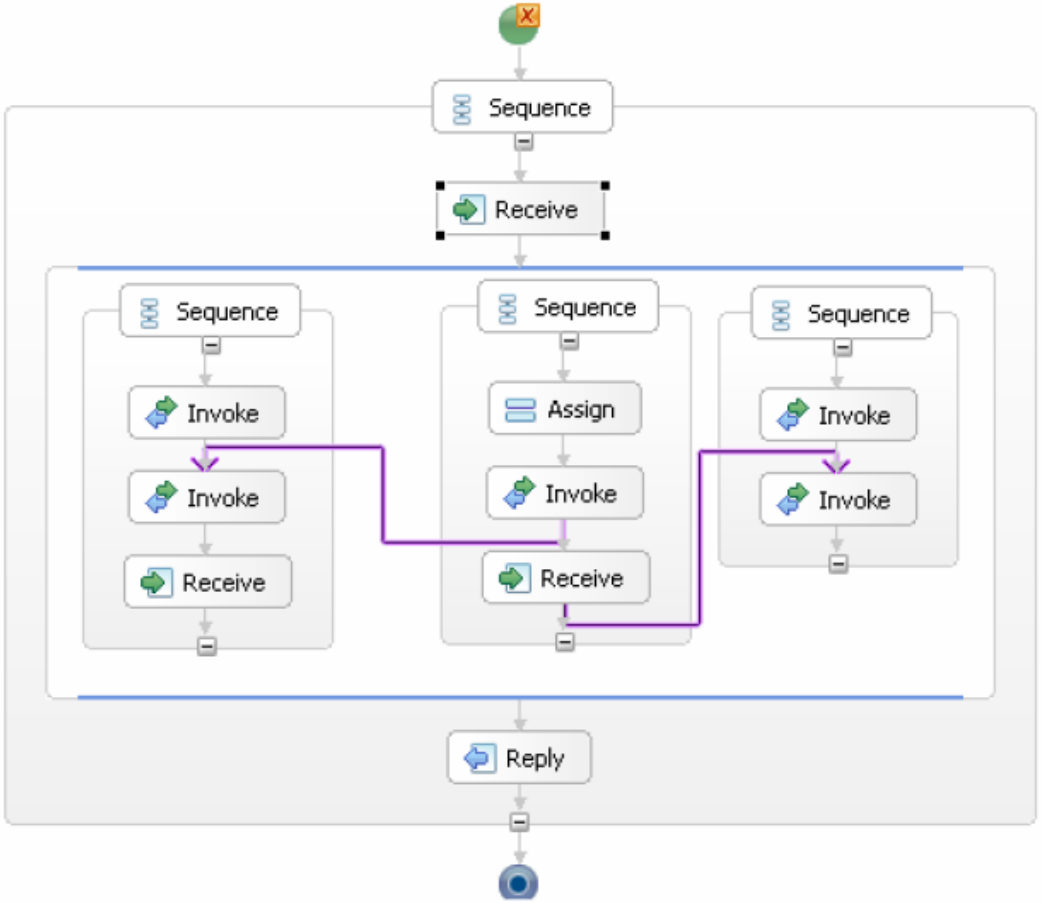


BPEL

- BPEL = Business Process Execution Language
- An XML-based **workflow language**
- **BPEL is the primary industry standard for** composing Web services
- View Web services as components from which to build more complex applications
  - –BPEL assumes all components are Web

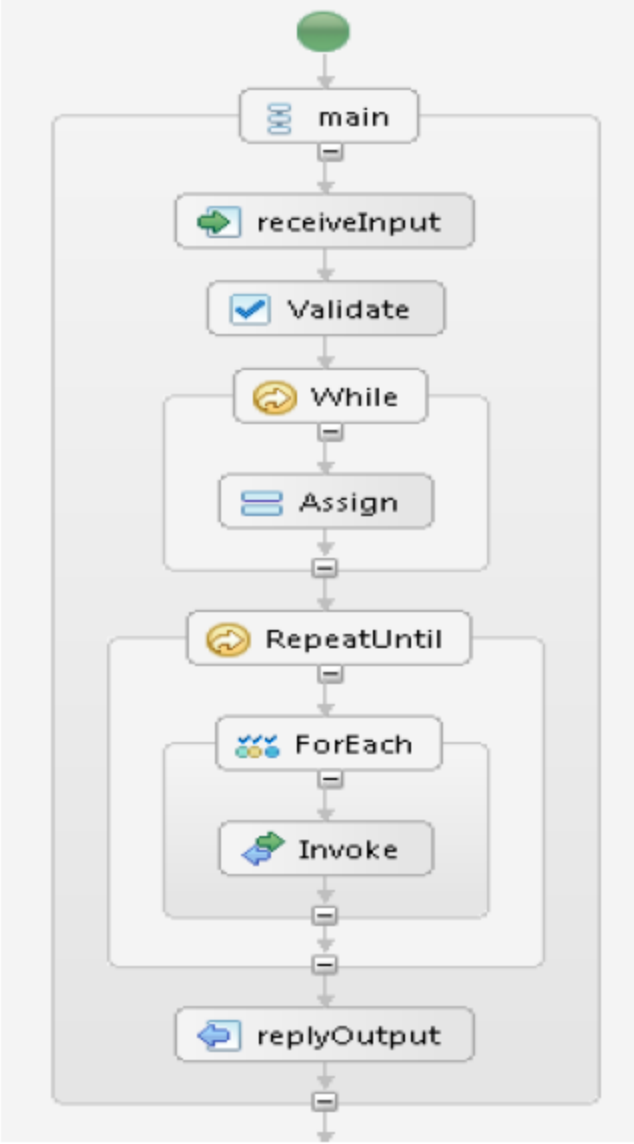
ActiveBPEL

- A BPEL engine which also incorporates a graphical modeling tool



Key features

- While
- If
- RepeatUntil
- For
- ……
- BPEL is a Turing-complete language



BPEL

- **Disadvantage**: complexity of the language

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
BPEL Process Definition
Edited using ActiveBPEL(tm) Designer Version 3.0.0 (http://www.active-endpoints.com)
-->
<process xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable" xmlns:bpel="http://docs.oasis-
open.org/wsbpel/2.0/process/executable" xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-
process/" xmlns:ns1="http://docs.active-
endpoints.com/activebpel/sample/wsdl/loan_approval/2006/09/loan_approval.wsdl" xmlns:ns2="http://docs.active-
endpoints.com/activebpel/sample/wsdl/invoke_with_catch/2006/09/invoke_with_catch.wsdl"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="InvokeWithCatch" suppressJoinFailure="yes"
targetNamespace="http://docs.active-
endpoints.com/activebpel/sample/bpel/invoke_with_catch/2006/09/invoke_with_catch.bpel">
  <import importType="http://schemas.xmlsoap.org/wsdl/"
location="project:/BPEL_Samples/Resources/WSDL/loan_approval.wsdl" namespace="http://docs.active-
endpoints.com/activebpel/sample/wsdl/loan_approval/2006/09/loan_approval.wsdl"/>
  <import importType="http://schemas.xmlsoap.org/wsdl/"
location="project:/BPEL_Samples/Resources/WSDL/invoke_with_catch.wsdl" namespace="http://docs.active-
endpoints.com/activebpel/sample/wsdl/invoke_with_catch/2006/09/invoke_with_catch.wsdl"/>
  <partnerLinks>
    <partnerLink myRole="assessor" name="riskAssessmentLinkType"
partnerLinkType="ns1:riskAssessmentLinkType"/>
    <partnerLink name="smallLoanApprovalLinkType" partnerLinkType="ns2:smallLoanApprovalLinkType"
partnerRole="approver"/>
  </partnerLinks>
```

```
<variables>
    <variable messageType="ns1:creditInformationMessage" name="creditInformationMessage"/>
    <variable messageType="ns1:riskAssessmentMessage" name="riskAssessmentMessage"/>
    <variable messageType="ns1:approvalMessage" name="approvalMessage"/>
  </variables>
  <sequence>
    <receive createInstance="yes" operation="check" partnerLink="riskAssessmentLinkType"
portType="ns1:riskAssessmentPT" variable="creditInformationMessage"/>
    <scope>
      <faultHandlers>
        <catch faultMessageType="ns1:errorMessage" faultName="ns2:loanProcessFault" faultVariable="V1">
          <assign>
            <copy>
              <from>concat('Approver Error! Error code: ', $V1.errorCode)</from>
              <to part="accept" variable="approvalMessage"/>
            </copy>
          </assign>
        </catch>
      </faultHandlers>
      <invoke inputVariable="creditInformationMessage" operation="approveSmall"
outputVariable="approvalMessage" partnerLink="smallLoanApprovalLinkType"
portType="ns2:smallLoanApprovalPT"/>
    </scope>
    <assign>
      <copy>
        <from part="accept" variable="approvalMessage"/>
        <to part="level" variable="riskAssessmentMessage"/>
      </copy>
    </assign>
    <reply operation="check" partnerLink="riskAssessmentLinkType" portType="ns1:riskAssessmentPT"
variable="riskAssessmentMessage"/>
  </sequence>
</process>
```

Exam's quizes

- **1.** Care este principiul de planificare folosit de algoritmul MCP?
- **2.** Descrieţi pe scurt modul de funcţionare al BPEL.