**DIFLOW: A DISTRIBUTED WORKFLOW MANAGEMENT SYSTEM**

by

ANUJ SUNIL SHETYE

(Under the Direction of Krzysztof J. Kochut)

## ABSTRACT

Workflow systems are one of the key technologies enabling automation of business processes and, recently, scientific applications. Traditionally, control of the execution of workflow processes has been centralized, despite the fact that they have frequently involved and coordinated systems executing at distributed computing nodes. Today, there is a need for decentralized and distributed workflow management systems (WfMS). In this thesis, we present DIFLOW, a system for designing and executing workflow processes based on dynamic migration of workflow instances during runtime. The system allows a process designer to define process constraints, which are specified in terms of process variables and capabilities of the workflow's processing nodes (performers). At runtime, workflow instances may migrate to computing nodes that satisfy the defined constraints. Process constraints in DIFLOW may capture functional or non-functional requirements of the process, which cannot be expressed using typical process definition languages, such as BPMN. In this thesis, we introduce a Constraint Definition Language (CDL) to describe constraints comprising of performer capabilities and domain specific variables for providing necessary migration meta-information. We also present a design and implementation of DIFLOW capable of scheduling and enacting workflow instances in a distributed environment.

INDEX WORDS: Distributed Workflow Management System, BPMN 2.0, Activiti

# DIFLOW: A DISTRIBUTED WORKFLOW MANAGEMENT SYSTEM

by

ANUJ SUNIL SHETYE

BE, University of Mumbai, India, 2010

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment

of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2014

**DIFLOW: A DISTRIBUTED WORKFLOW MANAGEMENT SYSTEM**

by

ANUJ SUNIL SHETYE

| | |
|---|---|
| Major Professor: | Krzysztof J. Kochut |
| Committee: | John Miller<br>Lakshmish Ramaswamy |

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
May 2014

## DEDICATION

I dedicate this work to my parents, brother and all my friends for providing me constant support and motivation.

## ACKNOWLEDGEMENTS

I would like to thank Dr. Kochut for his guidance, not only in this project, but also in my overall graduate academic career. I would also thank Shasha Liu (Amy), a Ph.d student at The University of Georgia for her valuable inputs in developing the system. Lastly, I would like to thank all my professors from whom I have learnt a lot throughout my academic career.

# TABLE OF CONTENTS

Page

# LIST OF TABLES

Page

## LIST OF FIGURES

Page

# CHAPTER 1

# INTRODUCTION

The emergence of grid technologies and cloud computing paradigm has enabled enterprises and various scientific communities to move the processing of their applications to external resources in order to take advantage of scalable, efficient, secure and cost effective architecture of these technologies. These application involve a large amount of data transfer, and control flows within interdependent components. Over the past few decades workflow systems have been used to represent these applications at a more abstract or a higher-level perspective and customizing these systems to execute in a decentralized way has long been an active area of research. According to the Workflow Management Coalition (WfMC), a workflow is defined as "the computerized facilitation of a business process in whole or part [29]. It is also be described as an orchestrated and repeatable pattern of business activity enabled by the systematic organization of resources into processes that transform materials, provide services, or process information [33]. Today, workflows or processes have become an integral part of most of the enterprises in various domains ranging from finance, manufacturing to human resources and span a number of participants and departments within or across enterprises. As Business Process Re-engineering (BPR) happens, the process keeps evolving which increases collaboration between more participants across departments and has necessitated the need to execute the workflows in a distributed environment  and due to Business Process Outsourcing (BPO) it has become more beneficial if the execution of the outsourced component takes place at the vendors site. Similarly, scientific workflow applications are a special type of workflow applications which comprise of a

series of computational or data manipulation steps in a scientific application and development of these applications have enabled scientists to perform advanced scientific experiments and large data processing. These applications constitute of often hundreds and thousands of tasks running in parallel and may involve implicit and explicit data and control dependencies. Therefore, the use of distributed technologies to translate these workflows on remote resources to execute more efficiently is always desirable.

The main challenge towards building distributed workflow management systems is to provide some type of meta information which provides functional and non-functional requirements (NFR) in the process definition during its design and deployment phase and enforcing them during the runtime of process instances. Some of the earlier work towards the approach involve partitioning of these processes based on role [14], by fragmenting loops and scopes [13], and provide partition using explicit data and control dependencies [12]. However, these works provide a static way of migrating the tasks where the partitions and hosts are specified at design time. Focus has been centered towards research regarding the migration of these workflows during runtime based on evaluation of some process level constraints and NFRs. Examples of these requirements are execution time, performance, capacity utilization for the execution. Also, in case of some scientific applications migrating the computation where experimental data is not transferable due to privacy and security reasons, in certain processes the requirement of computation of the tasks in a close geographical proximity where all expensive licenses and proprietary software are installed. To keep the approach simple, the declaration of these requirements must be as abstract as possible for the designer with only domain knowledge and not the actual implementation of the services. On the other hand Workflow Management Systems (WfMS) responsible for the execution and management of these processes should be

equipped with the functionality to evaluate these requirement mappings at runtime and take the necessary decisions for migrating the executions. In this thesis, we propose DIFLOW , a distributed workflow management system to enact distributed execution of workflow processes by evaluating each step which may be bound by a constraint expression composed of certain process level requirements and system performance capabilities [42].

The outline of the rest of the thesis is as follows.

Chapter 2 provides some background information on workflow management systems, BPMN 2.0 Activiti BPM and Object Constraint Language (OCL).

Chapter 3 discusses motivation and objective behind the approach proposed in the thesis.

Chapter 4 describes some earlier and existing work done on distributed process executions.

Chapter 5 gives a detailed description of the capability modeling and the syntax of the constraint definition language.

Chapter 6 describes the system characteristics and components.

Chapter 7 explains an overall architecture and implementation details of the proposed system.

Chapter 8 describes the evaluation of the system based on some case studies.

Chapter 9 concludes the discussion with a brief summary and the future work.

# CHAPTER 2

# BACKGROUND

## 2.1 Workflow Management System .

A Workflow Management System (WfMS) can be defined as a system that completely defines, manages and executes workflows through the execution of software whose order of execution is driven by a computer representation of the workflow logic [10]. It can also be seen as the underlying middleware consisting of concepts like relational database management, communication infrastructure, storage essential in choreography and orchestrations of a workflow process model. Though it is sometimes confused as an application, a user starts to execute the components of a process [23], some of the developed WfMS provide the necessary infrastructure to create and execute Web based forms required to execute these tasks. We further describe a WfMS as described in WfMC, the workflow reference model [10].

### 2.1.1 Workflow Management System Components

The system comprises of majorly two components Build time and Run time, where the build time functions are concerned with designing and deploying of the workflow models and run-time functions addresses the actual execution and management of workflows.

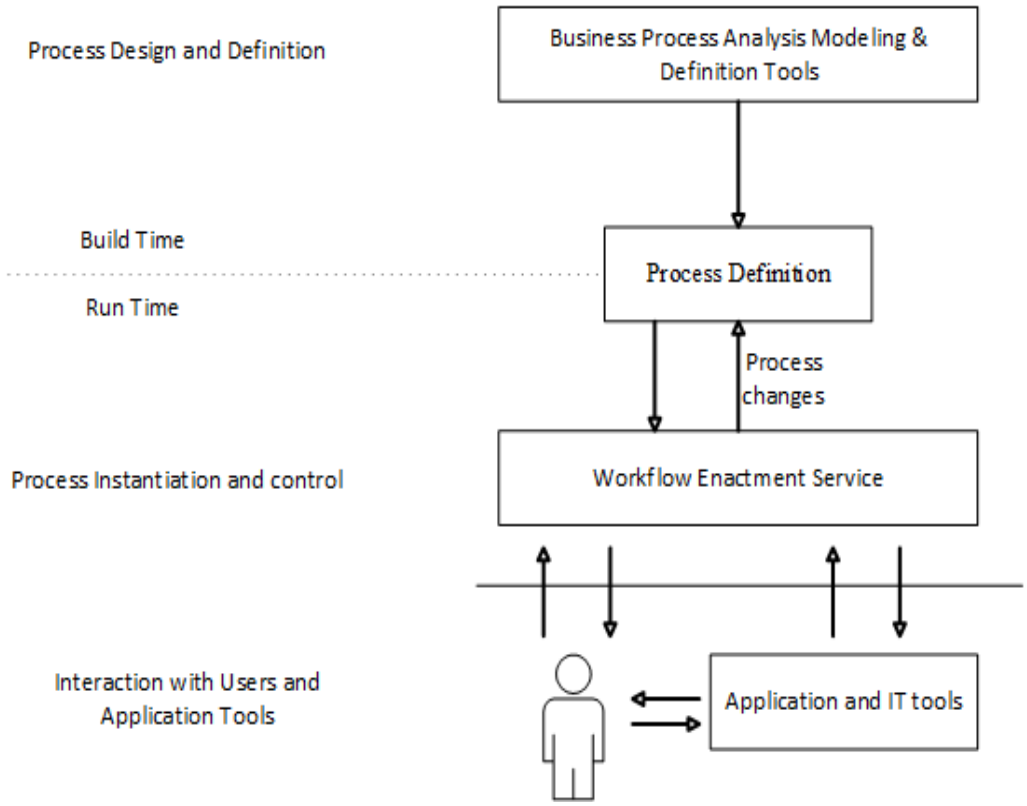


Figure 2.1: Workflow system components [10].

The figure 2.1 shows an abstract overview of a workflow system characteristics.

**Build-time functions** are those during which the actual designing of a process definition takes place and also the deployment of the abstract process model using a computer recognizable machine representation onto the system is performed. A process definition can be designed using formal notation using multiple third party vendors and sometimes, WfMS own designer interface. This is the phase where all the process participants and requirements mapping are performed.

**Run-time functions** provide the actual runtime mappings for data and control dependencies, and components to the required resources. As rightly stated "Run-time process control functions act as the linkage between the process as modeled within the process definition and the process as it

is seen in the real world, reflected in the runtime interactions of users and IT application tools." [10]. Newer WfMS systems have introduced many improvements, where a number of services are provided to manage the actual execution of workflows such as instantiating and suspending a process. Many systems also provide tools for auditing and reporting the execution of these instances.

Other components such as **Database** and support for **IT tools and resources** often are overlooked but are an essential part of the whole system. A relational database is used to persist the process definition and most of the temporal runtime information of the instances.

### 2.1.2 Workflow system structure

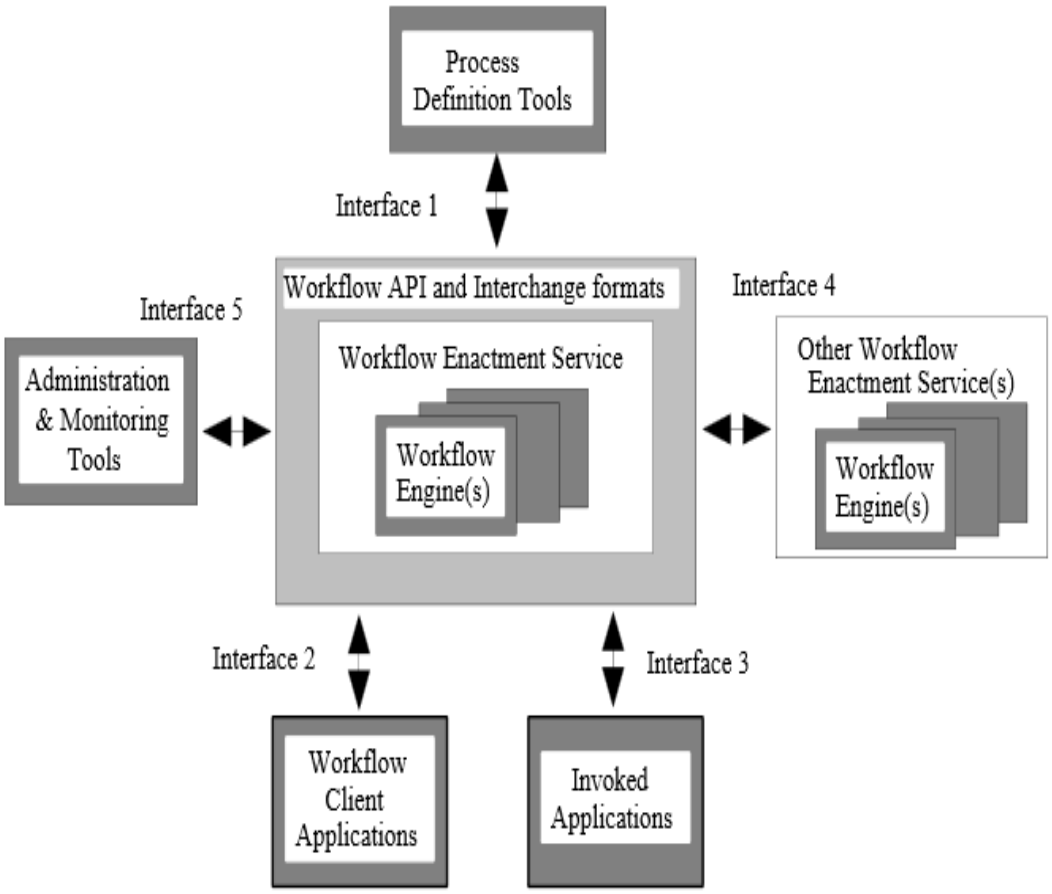


Figure 2.2: Workflow reference model [10]

Figure 2.2 shows the overall system structure of a WfMS. The main component is the Workflow Enactment service which consists of Process Engine(s) and is responsible for interpreting the process descriptions in executable format. The enactment service also handles the control flow information and overlooks the sequencing of activities in the order of execution. This is done with the help of the underlying process engine. The other interfaces are supplied to the enactment service via an Application Programming Interface (API).

The execution of the process definition in many of the WfMS is based on a rigorous mathematical model, such as Petri Nets [22] and the lifecycle of an instance can be viewed as a state machine diagram which can be seen in figure 2.3.

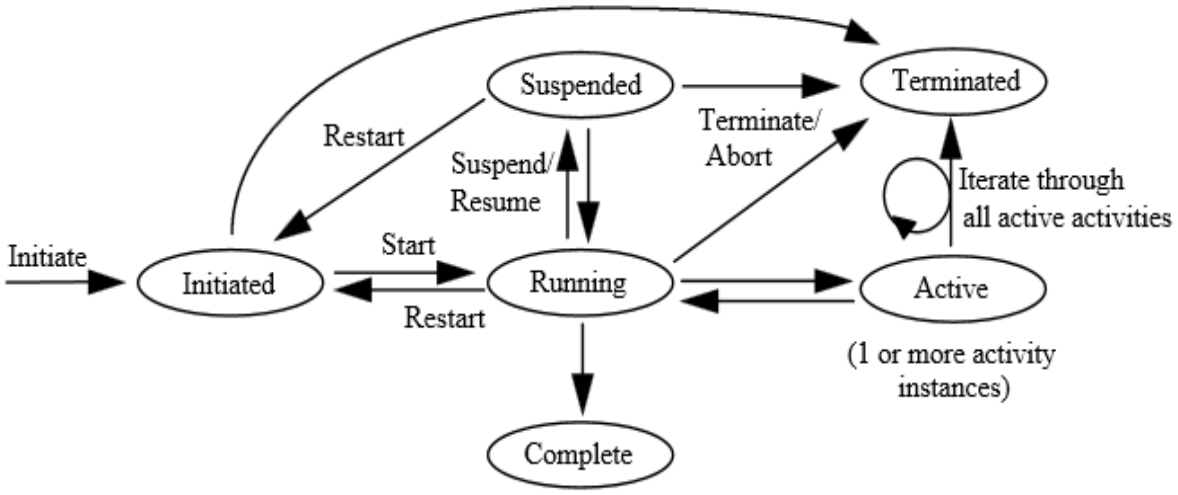


Figure 2.3: Process instance lifecycle [10]

The basic states in the cycle are:

**initiated** - a instance for a process definition has be created and ready for execution.

**running -** the execution of the instance is in progress.

**active -** an activity in an instance may or may not be running or maybe waiting for an external event.

**suspended -** the instance is suspended and execution cannot resume until its activated again.

**completed -** the execution has completed successfully

**terminated -** the instance may have been abruptly terminated due to some unforeseen conditions.

Most of the workflow management systems today are designed on the basis of above standards but the functionality provided by each system may vary for the type of systems they are designed for. The major focus of this work is to build a distributed management interface on top of such an existing Workflow Management System.

### 2.2 BPMN 2.0

Business Process Model and Notation (sometimes referenced to as Business Process Modeling Notation) is a graphical representation designed to specify business processes and is maintained by the Object Management Group (OMG) [21] since 2005. The latest version of BPMN is 2.0. Recently, the use of BPMN as a standard has gained importance and has started being widely used as an industry standard. It can be said that BPMN was introduced to enhance the design of process definition in a visual way as compared to the already existing XML based process definition languages like BPEL and XPDL. Since BPEL is still a widely accepted language for process execution and many of the WfMS provide BPEL compatible engines, BPMN designed process have to be converted to BPEL representation at execution time.

The designing of components of a process in BPMN mostly resembles a computer flowchart or an Activity diagram in Unified Modeling Language (UML). BPMN 2.0 comprises of range of

constructs. Figure 2.4 represents some of the basic set of elements which are used to model a process.

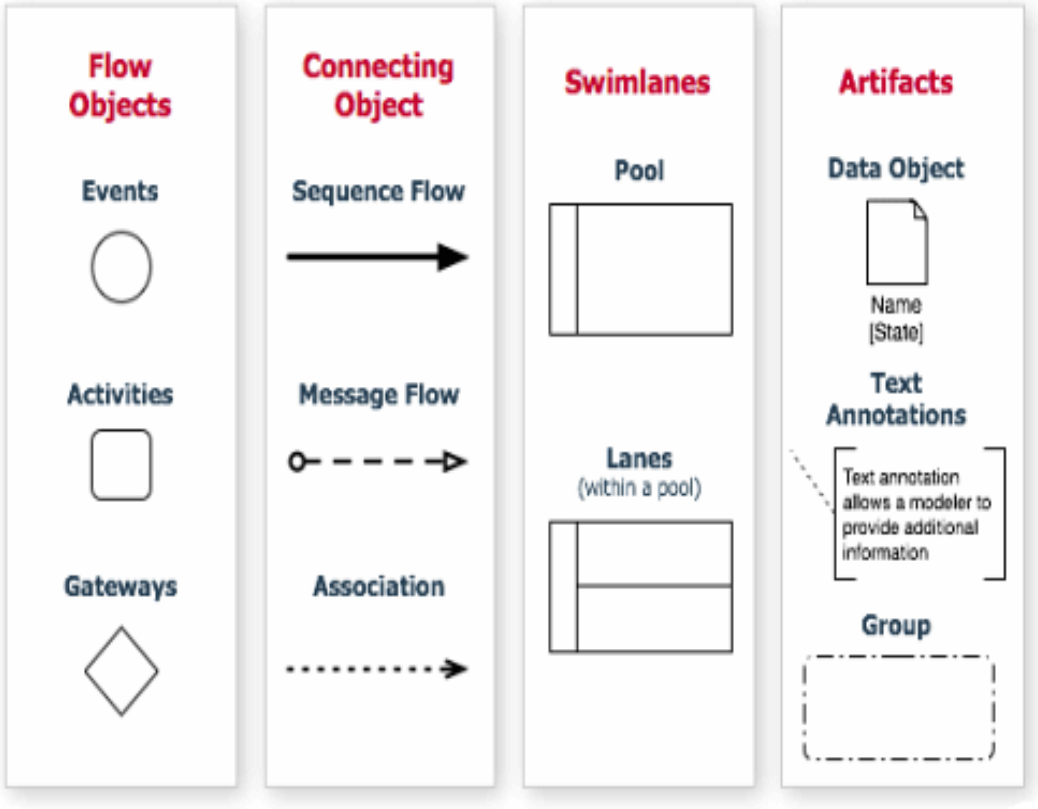


Figure 2.4. Basic BPMN 2.0 constructs

**Events** - Apart from start and end events BPMN provides compatibility for various throwing and catching events like error, signal, message, notification. These events could also be specified as boundary events.

**Activity** - Two major types of activities are User or Manual tasks which require human intervention and Automatic or Service tasks which are basically run using a computer program. There also are tasks for scripts, mail, web service and it is possible to extend BPMN to accommodate custom tasks for various behaviors.

**Sequence Flows and Associations** - These elements are used to represent the logical flow on the process. Apart from normal sequence flows there are message and data flows as well.

**Lanes, Pools and Artifacts** - BPMN has added ways to represent these components in the process to denote collaborations. These elements represent important information about the process without taking part in the actual execution and can be used to describe meta-information of the process. Apart from these elements, BPMN also adds extensions for sub processes both embedded and non embedded, loops, multi-instance support and many such elements.

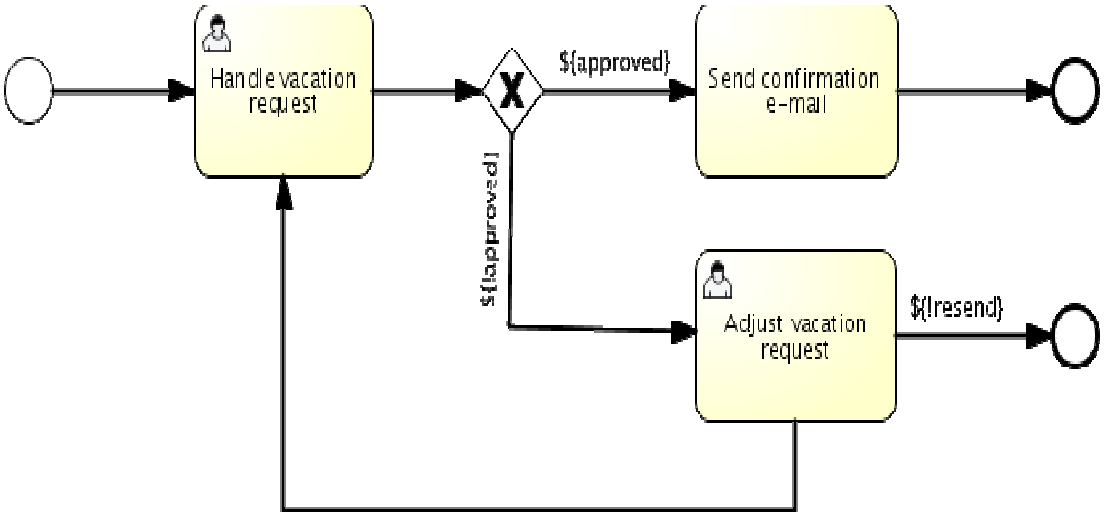


Figure 2.5: Vacation approval workflow [32]

As an example we describe a simple process designed in BPMN 2.0. The figure 2.5 shows the BPMN representation of the Vacation Request workflow. This can be created using any BPMN designer. The XML representation of this model is then used to map the constructs to the BPEL model. Figure 2.6 shows the XML representation of the Vacation Request workflow.

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions id="definitions"
          targetNamespace="http://activiti.org/bpmn20"
          xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:activiti="http://activiti.org/bpmn">

<process id="vacationRequest" name="Vacation request">
   <startEvent id="request" activiti:initiator="employeeName"></startEvent>

   <sequenceFlow id="flow1" sourceRef="request" targetRef="handleRequest" />

   <userTask id="handleRequest" name="Handle vacation request" />

   <sequenceFlow id="flow2" sourceRef="handleRequest" targetRef="requestAppDecision" />

    <exclusiveGateway id="requestAppDecision" name="Request approved?" />

   <sequenceFlow id="flow3" sourceRef="requestAppDecision" targetRef="sendApprovalMail">
     <conditionExpression >${vacationApproved == 'true'}</conditionExpression>
   </sequenceFlow>

   <task id="sendApprovalMail" name="Send confirmation e-mail" />

   <sequenceFlow id="flow4" sourceRef="sendApprovalMail" targetRef="theEnd1" />

   <endEvent id="theEnd1" />

   <sequenceFlow id="flow5" sourceRef="requestAppDecision" targetRef="adjustVacationRequest">
     <conditionExpression vacationApproved == 'false'} </conditionExpression>
   </sequenceFlow>

   <userTask id="adjustVacationRequest" name="Adjust vacation request"/>

   <sequenceFlow id="flow6" sourceRef="adjustVacationRequest"  targetRef="resendReqDecision" />

   <exclusiveGateway id="resendReqDecision" name="Resend request?" />

   <sequenceFlow id="flow7" sourceRef="resendRequestDecision" targetRef="handleRequest">
     <conditionExpression>${resendRequest == 'true'}</conditionExpression>
   </sequenceFlow>

   <sequenceFlow id="flow8" sourceRef="resendRequestDecision" targetRef="theEnd2">
     <conditionExpression >${resendRequest == 'false'}</conditionExpression>
   </sequenceFlow>

   <endEvent id="theEnd2" />
 </process>

</definitions>
```

Figure 2.6: XML representation of vacation approval workflow

The main question when use of BPMN is considered that if many of the WfMs provide BPEL [1] execution compatibility then BPEL should be used as the designing specification instead of mapping BPMN to BPEL for execution. Business Process Execution Language (BPEL) is a standard executable language for specifying actions within Executable and Abstract business processes with web services by extending the web service interaction model to support business transactions and by doing so, BPEL defines an interoperable integration model that facilitates the expansion of automated process integration in organizations. It has been a topic of debate whether BPMN and BPEL can co-exist on which Frank Leymann in [17] is able to clarify up to some extent. Due to fundamental differences in BPMN and BPEL it is difficult and in some cases impossible to generate BPEL readable code from BPMN. However, development of some systems like jBPM [37] and Activiti [32] provide a BPMN 2.0 compatible process engine which makes it feasible to run a BPMN process.

### 2.3 Activiti BPM

Activiti BPM or just Activiti [32] is a lightweight open source process engine designed to execute BPMN 2.0 business processes. It is distributed under the Apache license. It was started in 2010 by Tom Baeyens and Joram Barrez, two key developers of JBoss jBPM [37], a more mature open source workflow management system. They developed the system from their experience at jBPM and is built on a totally new code base. It is flexible to run as a standalone Java application, server, cluster or even cloud and can be easily integrated with the Spring framework [32]. Like jBPM, Activiti aims to provide a complete Business Process Management Solution, starting with the Activiti Designer to create a business processes using BPMN. The XML output of the Activiti Designer is deployed to the Activiti Engine that runs the process definition. The Activiti Engine executes automated steps, including invoking a web service, as

well as manual steps that involve people and Web forms [25]. The Figure 2.7 shows an abstract representation of Activiti BPM. Keeping in mind the continuous refinement of business processes and their models, Activiti developers attempted to provide flexible and robust functionality to their tool stack.

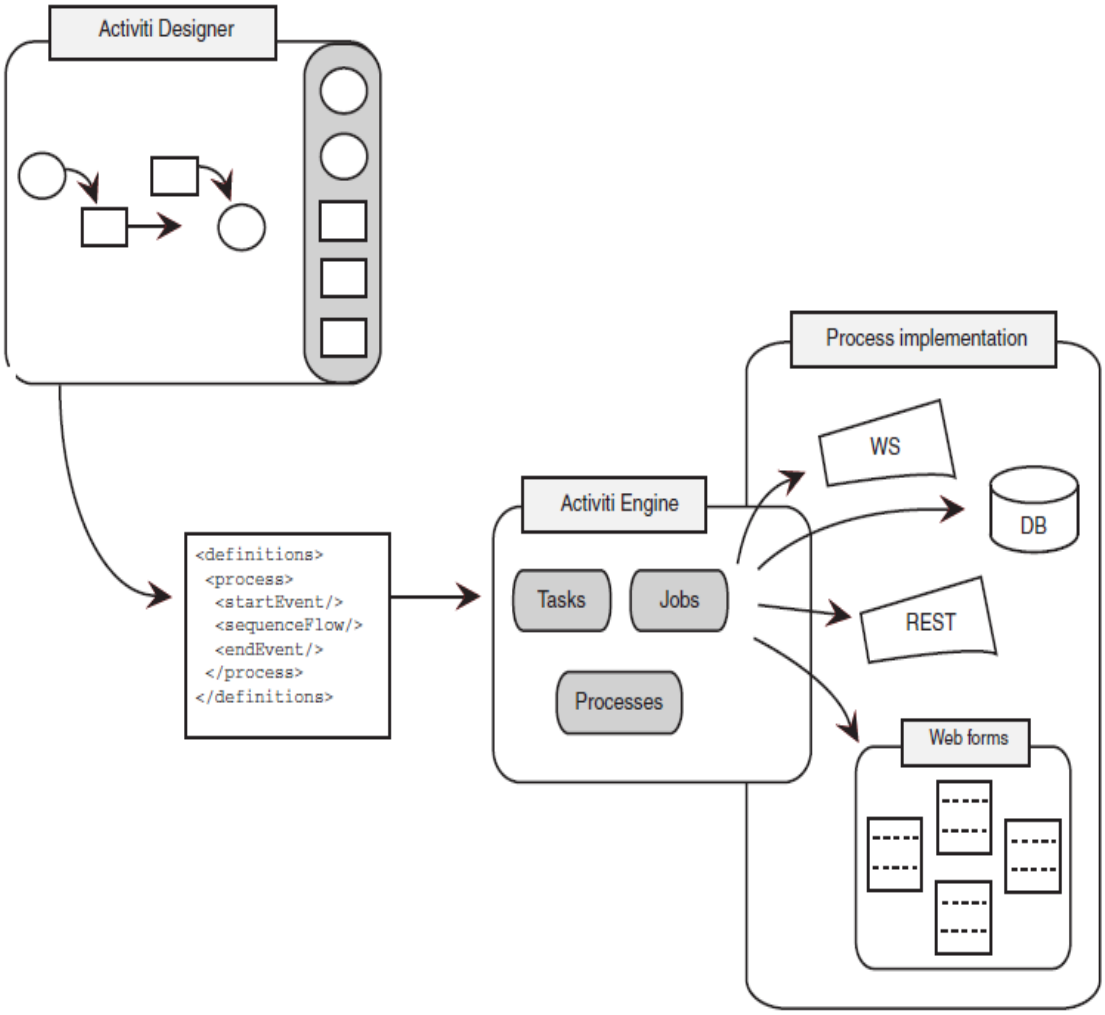


Figure 2.7: Abstract representation of Activiti

The core component of the Activiti framework is the process engine responsible for managing the execution of BPMN 2.0 processes and also many other things. Other components are built around the process engine. The other components as shown in Figure 2.8 comprise of

**Modeler**- A web based interface for designing processes developed based on KIS-BPM [39].

**Designer**- An Eclipse [34] plug-in for designing processes. Eclipse is an Integrated Development Environment (IDE) for developing Java projects.

**Explorer**- A web based application which provides an overall system management of Activiti.

**Rest**- A REST interface to the API and different engine services provided by Activiti

**Database**- A relational database forms the core of Activiti along with the process engine. The system heavily relies on the database for even smallest of the values ranging from the process definitions, process instances, jobs, process level variables and state information among others. Also the system is compatible with most of the available RDBMS in the market.

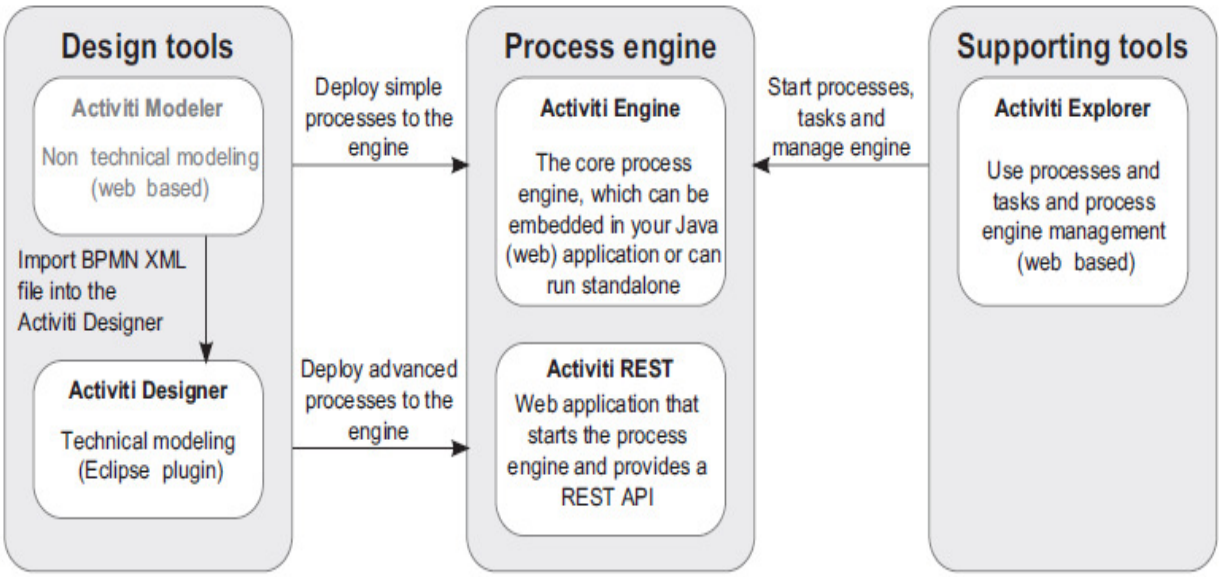


Figure 2.8: Activiti tool stack

At the lowest level, the working of Activiti can be looked as a simple state machine as shown in Figure 2.9. Thus, most of the BPMN 2.0 elements are implemented as a state and each state is

connected to incoming and outgoing flows. Every element can be attached with some execution logic that represent the actual behavior of that element.

The implementation level follows a very simplistic design making it flexible to add custom behavior to the system. Also the modular design of Activiti makes it easier to track the progress of the execution in the engine. Since it can be used in a distributed environment it is suitable for the work we present in this thesis.

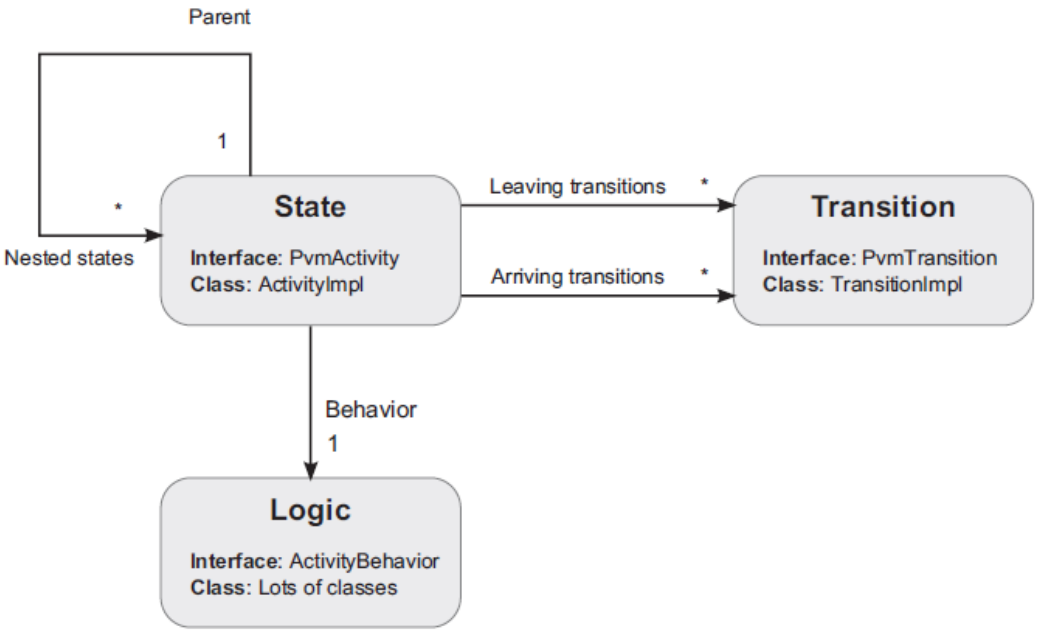


Figure 2.9: Activiti state machine

**2.4 Object Constraint Language (OCL).**

Object Constraint Language (OCL) [41] is a declarative language for describing rules that apply to Unified Modeling Language (UML) models developed at Rational and now included in the official specification of Object Management Group (OMG) meta model [27]. It can also be described as a textual language that provides constraints and object expressions which may not

be possible using visual notations[35]. A big advantage of OCL is as these expressions or constraints when added to a model does not change the actual logic of the context and are incapable of changing the state of the model and also has no side effects. However, an OCL constraint can be useful in constraining the actual execution of the context. It allows us to define four types of constraints namely pre, post, invariant and a guard condition. In this work we propose a Constraint Definition Language (CDL) based on OCL specification to constrain the execution of a process without modifying the actual process definition and the evaluation of these constraints during runtime to migrate the flow. We present a detailed description of CDL in later chapters.

# CHAPTER 3

# MOTIVATION AND OBJECTIVES

Traditionally, workflow process management systems have been designed to work in a centralized manner. These systems allow for distributing resources, such as user, machines and services, but the actual process control flow logic is executed by one single component at one single site [20]. Also the problem of multiple resource maintenance, synchronization overhead, large data transfers do not arise in these systems. However, as technologies and organizations evolve, the collaboration between different partners increases and workflow processes becoming more data and compute intensive, it has necessitated the development of decentralized workflow management systems. These systems are designed based on concepts of physical process fragmentation or modularization where the overall executable process is split into multiple subpart which are then distributed to different process engines for enactment. In contrast, decentralization can be achieved by process instance migration based on logical fragmentation such that only the responsibilities are distributed keeping the original structure intact. Such a migration would be a more natural way of executing a distributed process. Therefore, a process is described in subsequent steps which are passed from one machine to another, ensuring the task dependencies by sending tasks to their respective engines when all requisite conditions are satisfied [31]. This type of process fragmentation is possible on modern WfMS like JBPM [37] and Activiti [32] as these systems are database centric such that multiple process engines point towards a centralized database and can view the same process instance. However, the system

design is such that only one engine can execute the process instance at a time as they lack the necessary mechanism for validating and migrating during runtime.

Also, distributing process execution introduces interesting challenges. First, the process description should implement a formal meta model to describe the necessary requirements and conditions for the logic behind migration of the process and second the model should not modify the original execution logic of the process definition. Numerous approaches have been proposed to design such a data model. One of the approaches is the use of constraint modeling techniques in workflows [24]. Certain constraints can be specified on tasks which could be evaluated to decide the instance migration. Also, constraints should be added during the design time and should be mapped to the process executions at runtime. These constraints may be comprised of process level variables, various NFRs including quality of services, data requirements and location and temporal requirements among many other. To further highlight the use of this approach, we describe the following scenario.

GlycoQuant IDAWG$^{TM}$ workflow has been used by scientists at Complex Carbohydrates Research Center at The University of Georgia to perform quantitative glycomics analysis. Figure 3.1 shows an outline of the workflow. In short, IDAWG$^{TM}$ can be looked as a structure of four

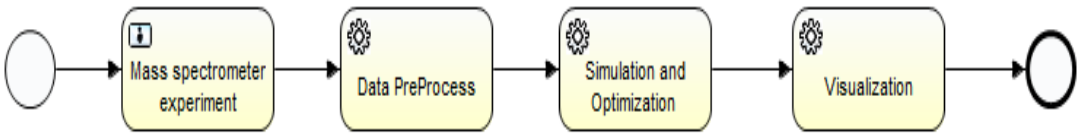


Figure 3.1: IDAWG$^{TM}$ process fragment

sequentially connected tasks. The raw data produced by the first task, Mass Spectrometer experiment is often large, ranging in size from a few megabytes to several gigabytes. This data undergoes a complex computational analysis (Simulation and Optimization), which consumes a

high amount of CPU cycles. Hence the data is typically transferred over the network to a remote high-performance systems. So, instead of transferring the data, if the Data Preprocessing task is transferred to the system were the raw data is residing, might prove to be an optimum solution. Another factor that makes computation migration more preferable to data migration is that the experimental data is obtained from long running tasks from running expensive experiments in that case scientists might not be willing to transfer the data due to security reasons. Hence, decisions can be made during runtime to migrate the process instances if the process is enriched with constraint expressions. Liu et al. present a similar approach in [18], which introduces a process constraint handling framework consisting of a Process Constraint Language (PCL) and a Process Constraint Ontology (ProCOnto). They address the problem of modeling and specifying domain specific constraints and non functional requirements in workflow processes and incorporating them within workflow applications.

In this thesis, we extend the work in [18] to create a process engine capable of handling process instance migrations and job execution scheduling based on decisions made by evaluating process constraints designed using a constraint definition language . Process level constraints and performer capabilities (QoS properties provided in the host machines) together form an expression. These expressions are then evaluated at runtime with process variables to select a suitable host for task execution. We have built our system on top of an existing open source system called Activiti [32].The main objective of this work is to extend the functionality of BPMN and Activiti to handle distributed processing of workflow activities by enforcing the constraints defined at design time to the process instance at runtime using custom signaling, validation mechanism and then scheduling and execution of these activities based on the decisions made after constraint evaluation.

# CHAPTER 4

# RELATED WORK

The chapter presents a discussion on some earlier and existing research work done on execution of workflows in a distributed environment and approaches of handling these executions by developing distributed workflow management systems. Executable workflow specification and workflow management systems have been an integral part of business processes which allow flexible and dynamic collaboration among several business partners. As stated in [31] "a single centralized system to control the execution of cross-organizational processes is often not technically nor organizationally desirable." Therefore as distributed process execution gains importance, many such approaches demonstrates the relevance of this research.

An extensively researched approach to address distributed workflow problem is to wrap the activities supposed to execute in a decentralized way with a web interface that whenever the tasks run they always run using the interface. A similar approach is proposed by Khalaf and Leymann in [14], where they present a system for role based business process fragmentation which can is specified by partner partitioning from the outset or partitioning the process after the selection of partner at a later time. The partitioning approach they follow is based on process control concepts such as loops, scopes and data dependencies presented by them in their previous work [12, 13]. Similarly, Baresi et al. in [2] propose a distributed orchestration of WS-BPEL processes using partitioning rules and fragmentation by creating corresponding invoke/receive activity pairs. However, these process fragmentation are defined during design time and have

been enforced in a static way. This might not be desirable in circumstances where the execution environment is dynamic and in case of most of the scientific applications.

These approaches look to address the issue of fragmentation at runtime. The work presented by Zaplata et al. in [31] proposes a process migration data meta-model which accompanies every process instance to support runtime migration of instances. They provide a unique approach to handle the parallel executions in process by introducing data class replication and execution to minimize over all synchronization overhead. The paper also address the security and privacy considerations using the process encryption. The work described in [28] presents a workflow process fragmentation and distributed execution method based on process mining. They build upon the idea of a single central management system to make the decisions of fragmenting the process at runtime based on the mined process properties. The migration decisions are made on the behavior of previous process execution.

Many research approaches have been aimed towards providing specifications for workflow processing in the grids and the authors of [3] propose a similar approach to provide high level composition of QoS-aware grid workflows. A notable contribution of this work is their XML based language for QoS Grid Workflow (QoWL) a subset of BPEL and a set of extensions used for specification of QoS requirements. In this work, they also address the issue of migration on grids based on location affinity. Another work addresses the issue of migration of workflow tasks to clouds [9]. Here, the migration is represented by modifying the actual process definition by adding two tasks, a task each at front for creating the service and at the end for destroying the service. The advantage in this approach is that the task is wrapped around an interface and for every execution an instance of the service is created at the computation site. The works presented above had the execution of business process as the main focus of their research. As in business

processes, distributing the execution of scientific workflow applications is highly desirable and in scientific workflow management systems, such as Kepler [19] and Taverna [11], which have sparked an interest into research for extending their functionality towards distributed frameworks. Pegasus [5] is a framework proposed by Deelman et al. for mapping complex scientific workflows onto distributed systems. They have built the whole system for an abstract workflow model for each fragment which is generated by making calculated selection of what they call as Execution horizon, data replication requirements and code migration. They have created their execution environments using DAGMan and CondorG [7, 8]. Some earlier work such as IntelliGEN [15] designed a distributed workflow system using METEOR [26] a scientific workflow management system and METUFlow [6] which provides a distributed scheduling mechanism, history and worklist management system. Also, Kochut et al. in [16] propose ORBWork, a CORBA based fully distributed, scalable workflow enactment service for METEOR workflow management system. They have proposed an enactment framework to support dynamic and adaptive workflows by providing a distributed scheduler, which provides necessary functionality to dynamically schedule tasks on different servers and oversee their execution. Yu and Bhuyya in [30] propose a taxonomy describing various constraint requirements at process level in a scientific workflow applications. The article [4] by Bhuyya et al. proposes a decentralized workflow management system for cloud platform.

In conclusion, the need for distributing the workflows is increasing in order to meet high data and computational requirements of the applications and to migrate the execution enforcing the dynamic provision and evaluation of the constraints during runtime is highly desirable. Most of the work described above is based on designing requirements and process level constraints using text based workflow models like WS-BPEL and XPDL specifications and the increasing

popularity of BPMN2.0 as a graphical based modeling tool for workflow specification has opened up new avenues for extending these formalizations to design more expressive extensions to provide meta information for dynamic distribution of process instances.

# CHAPTER 5

# PERFORMER CAPABILITY MODEL AND CONSTRAINT DEFINITION LANGUAGE

In this chapter we put forward a Performer Capability Model (PCM) used to describe the QoS properties, available resources and user defined capabilities of the host machines. We also introduce a Constraint Definition Language (CDL) used to describe constraints on the process tasks during designing phase of the process.

## 5.1 Performer Capabilities.

A performer can be described as a node (host machine) on the network capable of executing a workflow task. These performers are registered via an API provided in the system and are described in terms of capabilities. A capability is basically a key-value pair representing the system properties such as QoS requirements, size of RAM, CPU time, cost, resources required during execution, such as necessary code, data files and sometimes user defined properties, such as specific licenses for e.g. AcrobatPro for creation of PDF files, or access to genomic databases in bio-informatics applications. These values are stored in the database and are retrieved during the constraint evaluation phase to make the necessary decisions. Figure 5.1 shows a capability general format .

General format:

propName [ '=' propVal ]

Where :
propName: property name
propValue: value denoted by a literal. Also optional in some cases

Figure 5.1: Capability general format.

Some examples of performer capabilities are shown as follows:

- HostAddress

  e.g. , hostId = “12.192.4.5”

- License or Authentication certificates

  e.g., AcrobatPro, MSOfficePro

- Resource Availability

  e.g., InputFileAvail = “input.dat”

- QoS properties

  e.g., RAM = 8Ghz , CPU = 3.2Ghz etc

These capabilities can also be viewed as variables describing a host machine.

**5.2 Constraint Definition Language.**

During process design constraint expressions are added and mapped to the respective elements in the definition. To specify a standard representation of these mappings we have designed Constraint Definition Language (CDL). The syntax (CDL) is similar to that of Object Constraint Language [27] specification.

**5.2.1 Constraint Expressions**

Constraint expressions when evaluated during runtime evaluate to a Boolean value and consists of combinations of performer capabilities and process variables i.e. variables defined in the workflow process definitions to facilitate the execution of workflow instances. Table 5.1 shows an outline of the syntax of the Expression Language (defined by Extended Backus-Naur form). A primitive is the smallest expression and can refer to a process variable or a performer capability property. literals are treated as constants and can be denoted by string or a number. The expressions support all types of relational and logical operators including "and", "or" and "not" .

Table 5.1: Expression language syntax

| | | |
|---|---|---|
| expression | ::= | anded_exp { ‘or’ anded_exp } |
| anded_exp | ::= | not_exp {‘and’ not_exp} |
| not_exp | ::= | [‘not’] primary_exp |
| primary_exp | ::= | primitive \| primitive relop literal \| |
| | | primitive relop primitive \| |
| | | literal relop primitive \| ‘(‘ expression ‘)’ |
| primitive | ::= | proc_var \| capability_prop |
| literal | ::= | string \| number |
| relop | ::= | ‘==‘ \| ‘!=‘ \| ‘<=‘ \| ‘>=‘ \| ‘>’ \| ‘<‘ |

Table 5.2: Constraint declaration syntax

| | | |
|---|---|---|
| constraint_declaration | ::= | “constraint” name |
| | | activity_context |
| | | constraint_type expression |
| activity_context | ::= | “context” name {“,” name} |
| constraint_type | ::= | “pre:” \| “post:” \| “inv:” |
| name | ::= | string |

### 5.2.2 Constraint Declaration

Table 5.2 shows an outline of a constraint declaration. A constraint declaration is defined by keyword "constraint" followed by the name followed by context and the expressions.

| | |
|---|---|
| constraint | RequestApprover |
| context | HandleVacationRequest |
| inv: | approver != requester |

Figure 5.2: Constraint declaration Example 1

```
constraint      ProPdfLicenseRequired
context         DocumentGenerateTask
pre:            COST == 100 and inputAvail == "GeneFile.dat"
                and (AcroBatPro or NuancePdfPro)
```

Figure 5.3: Constraint declaration example 2

Figure 5.2 and figure 5.3 show two examples of a constraint declaration. Figure 5.2 describes a constraint RequestApprover on the Vacation Request process constraint so that the HandleVacationRequest task cannot be completed by the person who is requesting the vacation and this condition should be true before and after the execution of the task. Figure 5.3 describes a suitable constraint. The execution of automated document generated task is performed if and only if the input file and either of the pdf generating licenses are available. If the system evaluates to true it continues else it finds another host with the required resources to run the task.

### 5.2.3 Constraint Types

CDL allows the designer to use three types of constraints namely pre , post and invariant. The migration decisions are taken depending upon the type of constraints specified on the context.

Pre: is a type of constraint that must be true before the execution of the task has begun the evaluation Semantics of a pre: constraint is shown in figure 5.4

```
pre conditions:
        evaluate the constraint condition on the current host of the process instance
                if true, continue on the same host
                otherwise look for a host that satisfies the constraint condition
                        if found continue the instance on that host
                        otherwise raise a workflow error event with the same
                        name as the constraint
```

Figure 5.4: Pre condition evaluation semantics

similarly a Post: constraint type, shown in Figure 5.5, must be true after the execution of the task is performed. The evaluation semantics are similar to the pre: constraint evaluation but it is done after the task has completed execution. So is the Invariant (Inv:) constraint which should be true throughout the execution of the task. The evaluation semantics is the combination of Pre: and Post: constraints, as shown in figure 5.6.

In this chapter, we have introduced the overall syntax of the constraints used to describe process level requirements attached to the tasks during the process design phase.

```
post conditions:
        evaluate the constraint condition on the current host as the process instance
                if true, continue on the same host
                otherwise look for a host that satisfies the constraint condition
                        if found continue the instance on that host
                        otherwise raise a workflow error event with the same
                        name as the constraint
```

Figure 5.5: Post condition evaluation semantics

```
invariant conditions :
        evaluate pre condition before the execution of constrained task
        execute the constrained task
        evaluate post condition after the execution of constrained task
```

Figure 5.6: Invariant condition evaluation semantics

# CHAPTER 6

# DIFLOW SYSTEM

In this chapter, we describe the design of the distributed workflow management system. As stated earlier, we extend the functionality of Activiti to suit our needs, therefore we have developed all the components around the process engine and API provided by Activiti. Figure 6.1 shows an overall system components of the DIFLOW System.

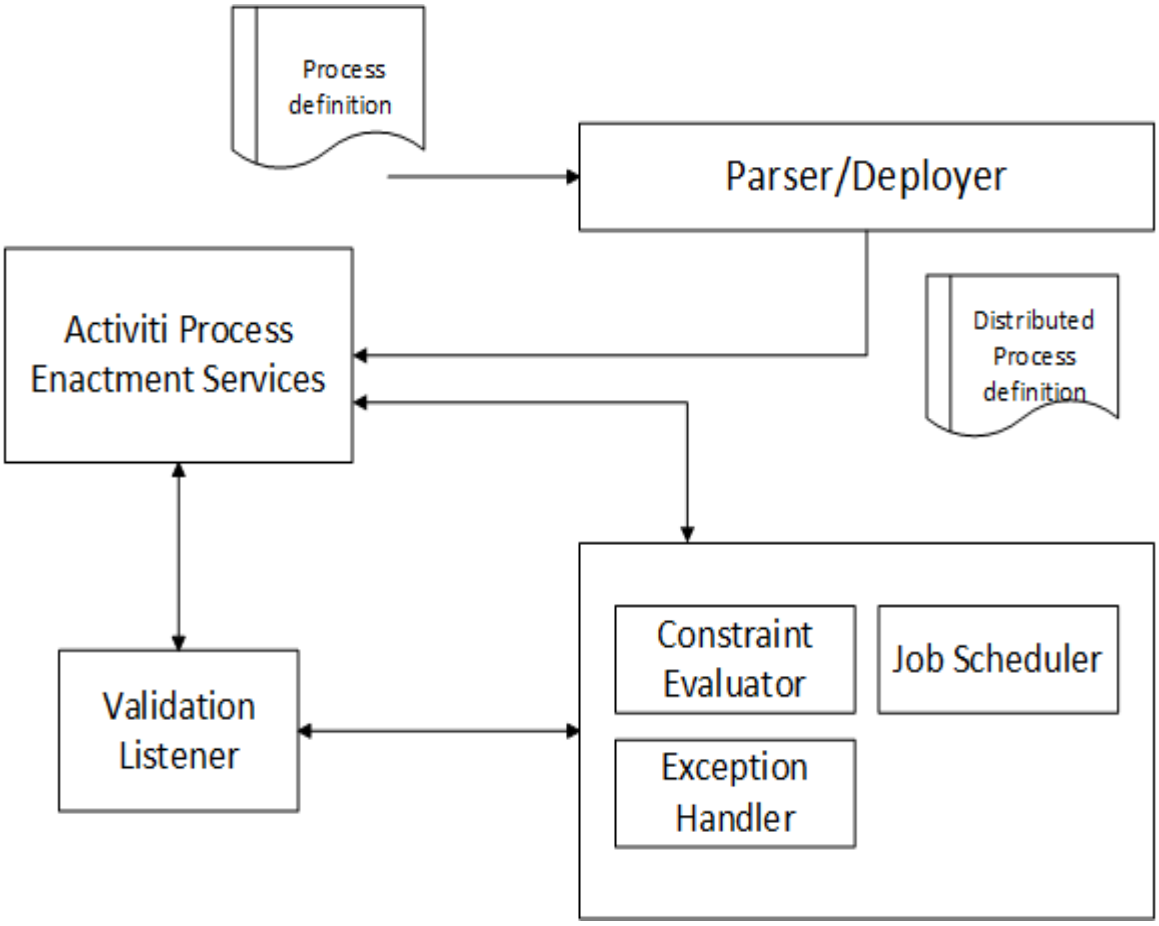


Figure 6.1: DIFLOW system components

### 6.1 Distributed Process Definition

During design time a process designer specifies the constraints discussed previously as an artifact in the definition. These artifacts represent only meta-information about the process definition and do not take part in the actual execution of the instances. Therefore, some pre-processing is required on the original process definition for specifying the mappings between the constraint and contexts. The preprocessed file is then deployed in the workflow management system. Before explaining the structure of the distributed process definition, we would like to discuss the execution of actual process instances in the Activiti engine and the approach taken towards the design of the translation scheme.

As discussed previously, Activiti can be looked upon as a state machine so all the process instances in the engine execute on concepts of a state. So, it is safe to say that one process instance execution is a completion of a number of states in a sequence. Therefore, the main challenge in migrating these states is a way to interrupt this execution on one machine and resuming on other machine. An execution can only be interrupted if a wait state is induced in a the system. Also, in Activiti, if the execution enters a wait state, the state of the execution is persisted into the database, as a checkpoint in case of a needed failure recovery. There are basically two major type of activities in BPMN: User tasks and Automated or Service tasks. Consider a process instance comprising of user tasks, as these task require human intervention they cannot be completed unless a user completes it. Hence such type of tasks induce a wait state in the system, but is not the same for a process instance comprising of automated tasks. As these tasks are automatic, Activiti is not able to induce a wait state in the system, thus interrupting this execution is not possible. Hence, to address this problem, a BPMN extension is introduced known as Asynchronous extension. If an activity is marked as asynchronous (from

here on we refer to it as  async extension) the system enters a wait state and can only be resumed if an external signal is received to restart it. Therefore the activities in DIFLOW on which the constraints are specified are marked as async. Figure 6.2 shows a fragment of IDAWG ™ process designed using the designer with artifacts specifying some constraints. Similarly, Figure 6.3 presents BPMN representation of the process definition after the preprocessing. In the example, it can be seen that an activiti:async attribute is added to the Data PreProcess and Simulation and Optimization tasks.

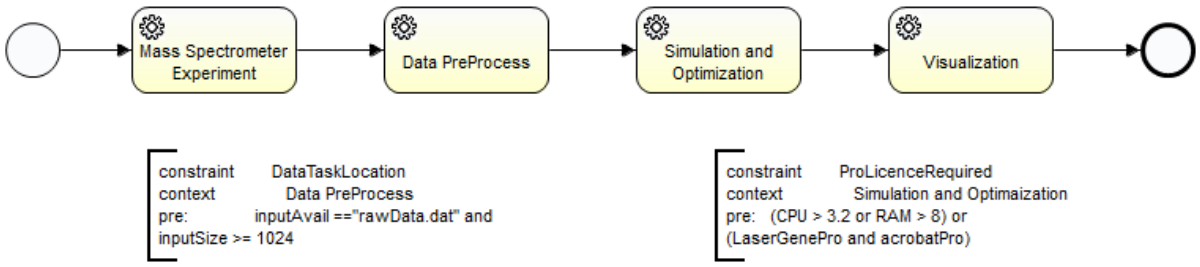


Figure 6.2: Distributed process definition

### 6.2 Parsing and Deploying

The parsing and deploying is done only at the design time. As shown in Figure 6.4 the process parsing is performed in two phases

**Phase 1** does the actual parsing of the XML file of the process and extracts the constraints from the artifacts. These artifacts can also contain comments hence only the well formed constraints described using CDL are considered. These constraints are then processed using a syntax parser and the result of the processing is a syntax tree for the expression. These syntax tree representations are stored in the database for the corresponding tasks and are retrieved at run time during the constraint evaluation. The syntax parser is designed using the context free

grammar specified for the Constraint Expression Language and uses recursive descent parsing for generating the  syntax tree. These syntax trees are evaluated during runtime for the referenced properties and capabilities using top-down approach. Figures 6.5 and 6.6 show abstract tree representations for constraint expressions shown in Figure 6.2.

```
<process id="myProcess" name="My process" isExecutable="true">

  <startEvent id="startevent1" name="Start"></startEvent>
  <sequenceFlow id="flow1" sourceRef="startevent1"
targetRef="usertask1"></sequenceFlow>
  <userTask id="usertask1" name="Mass spectrometer experiment"></userTask>
  <serviceTask id="task2" name="Data PreProcess" activiti:async="true"></serviceTask>
  <sequenceFlow id="flow2" sourceRef="usertask1" targetRef="servicetask2">
    <extensionElements>
       <activiti:executionListener event="take"
      class="org.activiti.ValidateInvoke.class"></activiti:executionListener>
    </extensionElements>

  </sequenceFlow>
  <serviceTask id="servicetask3" name="Simulation and Optimization"
activiti:async="true"></serviceTask>
  <sequenceFlow id="flow3" sourceRef="servicetask2" targetRef="servicetask3">
  <extensionElements>
      <activiti:executionListener event="take"
       class="org.activiti.ValidateInvoke.class"></activiti:executionListener>
      </extensionElements>
  </sequenceFlow>
  <serviceTask id="servicetask4" name="Visualization"></serviceTask>
  <sequenceFlow id="flow4" sourceRef="task3" targetRef="task4"></sequenceFlow>
  <endEvent id="endevent1" name="End"></endEvent>
    <sequenceFlow id="flow5" sourceRef="servicetask4"
targetRef="endevent1"></sequenceFlow>
  <textAnnotation id="textannotation1">
      <text>constraint       DataTaskLocation
            context          Data PreProcess
            pre:              inputAvail =="rawData.dat" and inputSize &gt;= 1Gb</text>
  </textAnnotation>
  <textAnnotation id="textannotation2">
      <text>constraint       ProLicenceRequired
            context          Simulation and Optimaization
            pre:   (CPU &gt; 3.2G or RAM &gt; 8G) or (LaserGenePro and acrobatPro)
</text>
    </textAnnotation>
 </process>
```

A

B

C

D

Figure 6.3: XML representation of distributed process definition

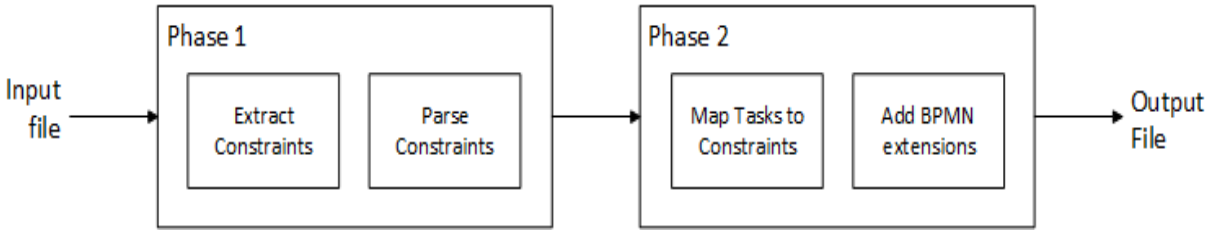


Figure 6.4: Parsing in DIFLOW.

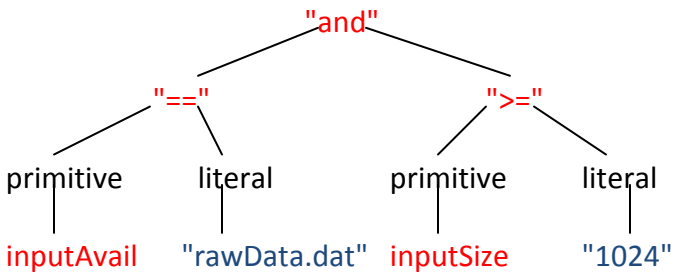


Figure 6.5: Parse tree for constraint 1

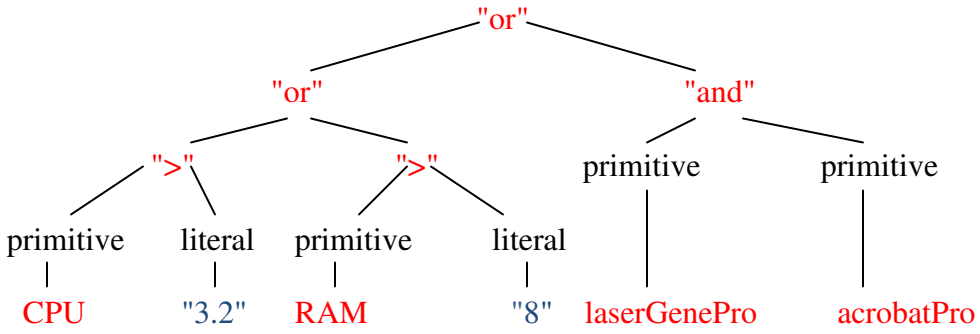


Figure 6.6: Parse tree for constraint 2

Once the extraction of constraints is done, **phase 2** of the parser does the actual mapping of these constraints to the tasks by storing the information in the database for the corresponding definition. Also, the necessary additions to the process definitions are done such as adding the async extension shown as A and B in Figure 6.3 and the necessary evaluation trigger mechanism depending upon the type of constraints is performed.

The result of this parsing step a .bpmn file with the necessary modifications is then deployed in the process engine using the interface provided by Activiti.

### 6.3 Validation Invoking Mechanism

During the runtime of a process instance when a task with constraint requirement is encountered, a module has to be executed in order to perform the evaluation and this module is called from within the instance. Hence the validation invoking mechanism can be seen as a way to tell the system that the current task is supposed to be evaluated for a constraint. This is done using a mechanism called as Execution Listener in Activiti. An execution listener can be seen as a piece of code invoked when an event occurs on the element such as start, end events. These execution listeners are shown in Figure 6.3 indicated by C and D

We design execution listeners each for the three different types of constraints, i.e. pre:, post: and inv: and the time they are invoked also are different for an instance if the constraint on the element is pre: condition then the execution listener is invoked at the end of the incoming sequence flow. Similarly, if the condition type is post: the execution listener is invoked at the end of task execution and if the condition is inv: a combination of both the execution listeners is used. The responsibility of these execution is to start the constraint evaluation modules and return the control to the process instance.

### 6.4 Constraint Evaluation

Constraint evaluation is triggered by the Validation Invoke call and is responsible for evaluating the previously described parse trees. Here the values in the syntax trees are substituted with the process variables and the performer capabilities. As the whole system is decentralized, each host has its own copy of constraint evaluation modules. First, the constraint is evaluated for the

current host and the execution continues if the evaluation succeeds, but if the constraint fails for that host, it then starts evaluating for other hosts one by one until a matching host is reached.

### 6.5 Exception Handling

If constraint evaluation for all of the deployment hosts fails then the execution cannot proceed and the process instance has to be terminated. Another approach is to invoke compensation module of execution when an error occurs inside the system. This flow can be a sub process which is activated as a consequence of the occurred exception and can be defined by the designer to handle the violated constraint. The exception handling mechanism of DIFLOW is unique such that when failed constraint exception occurs the actual executable code of the task is replaced by an exception handling code during the runtime. So that when the task executes an alternate control flow is invoked which handles the exception. The process designer is left with the decisions about the methods to handle the exceptions.

### 6.6 Job Scheduling

As soon as a host matches the constraint the job scheduler sends the task to the job queue of the host machine. Here the task or a job is picked up by the Job Executor in the Activiti process enactment service and forwards it for execution by the process engine. A Job Executor is not an actual executor it is a listening mechanism which maintains a thread pool. It checks the job queue for recently added jobs. If a job arrives at the queue, the job executor de-queues the job and starts a separate execution thread. Thus, at one time a process engine can execute multiple jobs.

# CHAPTER 7

# IMPLEMENTATION

## 7.1 Architecture

The architecture of the system comprises of two major components one for process designing interface and the other for scheduling and execution management. The overall architecture can be seen in figure 7.1. **Process design and management interface** is the user interface for interaction with the system.

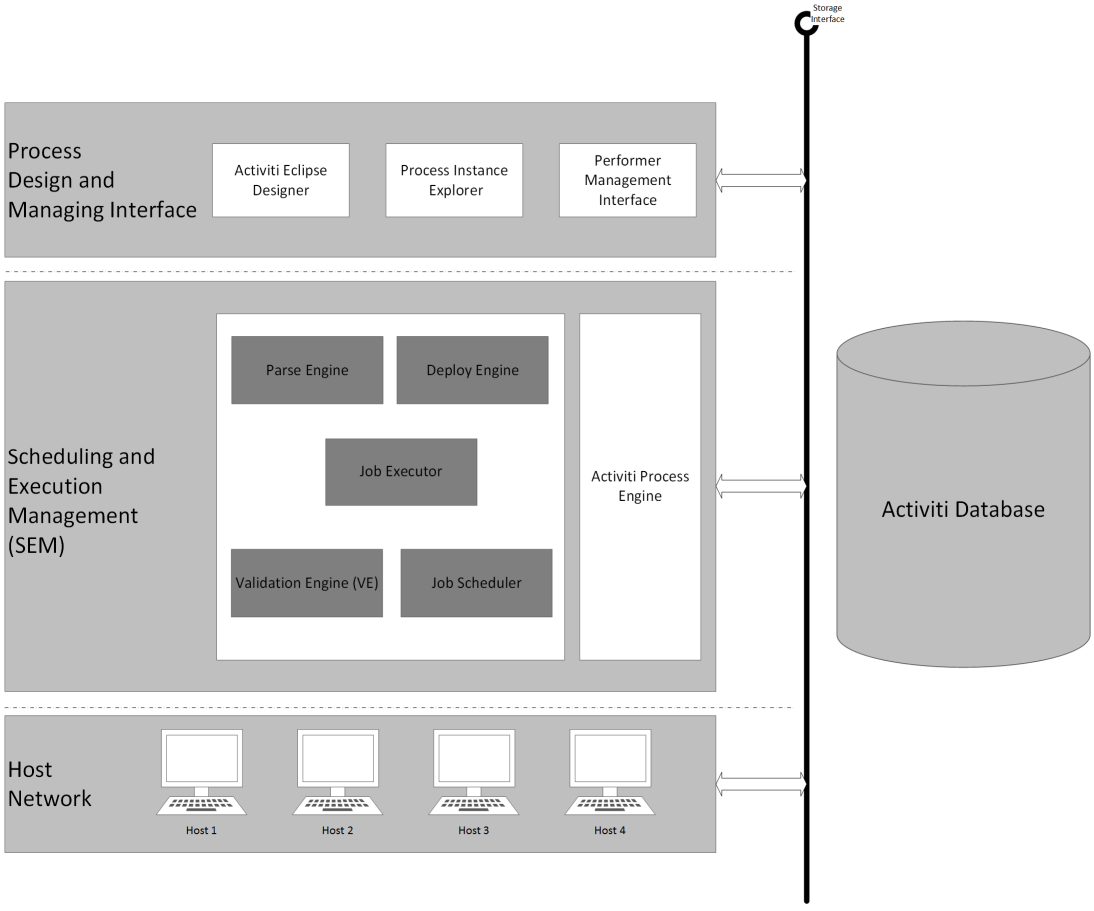


Figure 7.1: System architecture

**Activiti Eclipse Designer** is an eclipse based plug-in which enables a developer to create a BPMN 2.0 process that can be executed in the process engine. It can also be used to import existing processes definitions, create test cases and build deployment artifacts (code for running instances, process images etc) . Figure 7.2 shows a screen shot of the designer. In addition to that Activiti provides another web based designer was developed using KISBPM [39].

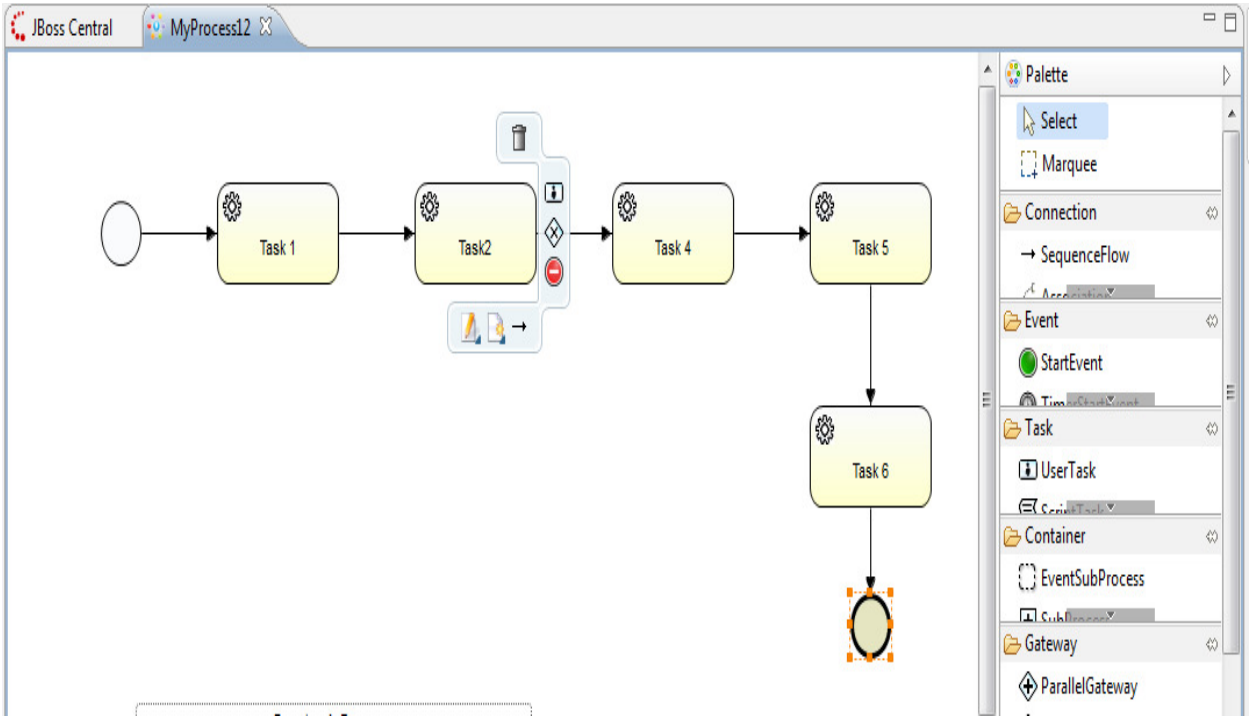


Figure 7.2: Activiti Eclipse designer

Also Performers (Host machines) can be described using a web based **Performer management interface**. It is provided as a web application, and can be used to register the performers with DIFLOW, we also provide functionality to add, delete and edit capabilities associated with each host. Figures 7.3 and 7.4 show screenshots of the performer management systems.

One of the main contribution of our work is the **Scheduling and Execution Management** (SEM) and the other being the Constraint Definition Language. The whole system is designed to

work in a decentralized way such that each host has its own copy of the execution and is capable of evaluating and scheduling the tasks by itself.

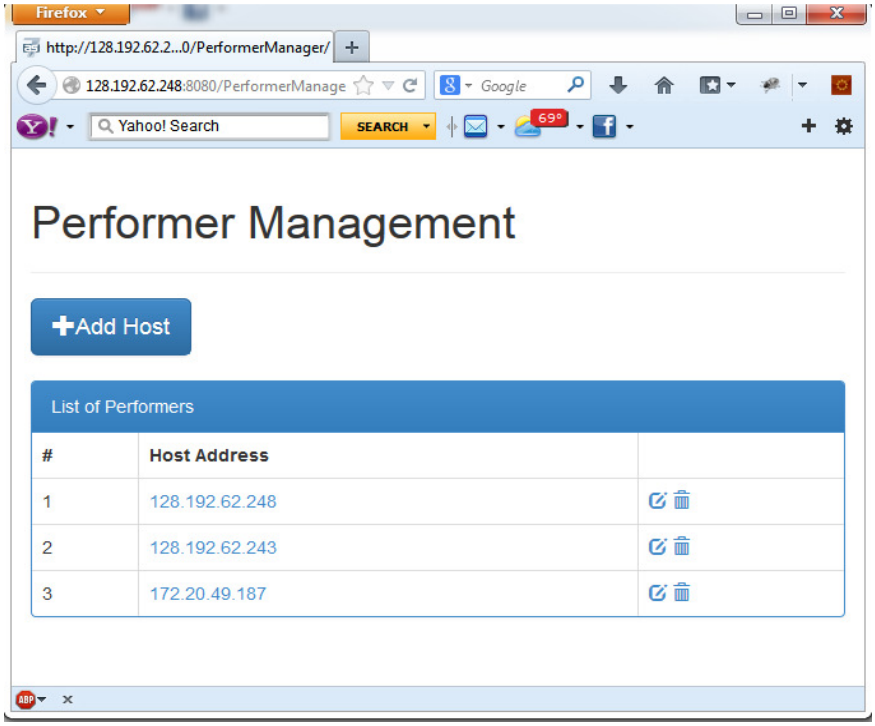


Figure 7.3: Performer Manager Screen 1

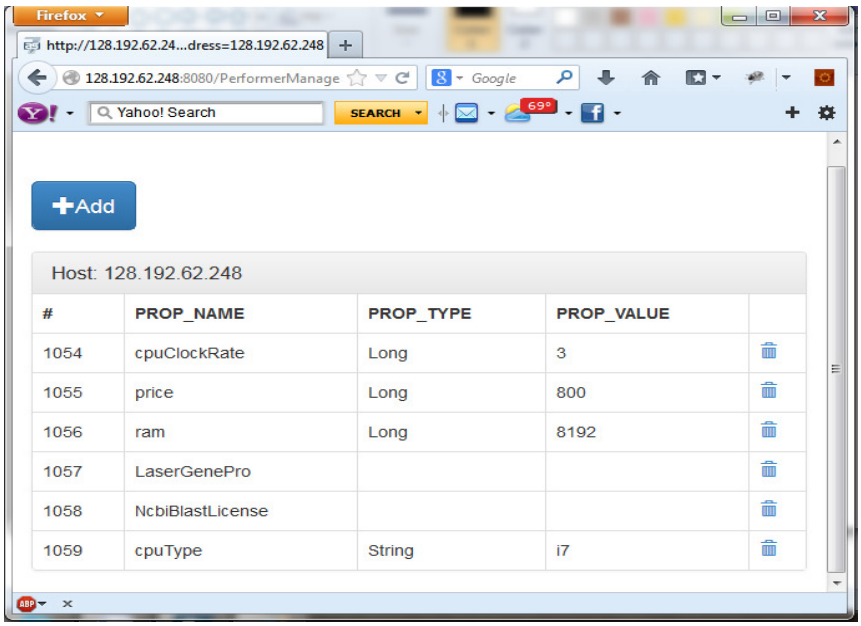


Figure 7.4: Performer Manager Screen 2

The components of SEM are as follows.

**Parse engine** and **Deploy engine** are used to parse the BPMN process file and perform necessary modifications and deploy using the Activiti API for the deployment of the system. One thing to consider here is the original BPMN file is also Activiti compatible and can be deployed directly without parsing it. However, this time process definition will be created without any annotations on the system. An instance can be started, but it will execute in a centralized way. **Validation Engine** provides the functionality for evaluating the constraints and **Job Scheduler** just puts the job in the job queue of the target machine. The important approach in this system is the way a **Job Executor** is designed. A job executor is not an executor but is a mechanism to schedule the execution of the job on to the process engine. It is a thread listening for new jobs added to the job queue. When a job arrives, it de-queues the job and starts an execution thread which in turn is processed by the Activiti Engine, responsible for completing the tasks. These threads are managed by a Thread pool to manage the number of threads generated in the system.

### 7.2 Prototype Implementation

Based on the system architecture the DIFLOW system has been implemented as a Java web application which is run via an application server such as JBoss [38]. First the idea was to implement a single manager which would be responsible for evaluating and scheduling the jobs but we realized to attain a robust management of the system it would be beneficial to have a decentralized system, such that each host machine is capable of deciding where the next task is executed. The system is build upon the underlying idea of running multiple process engines against one centralized database. DIFLOW is implemented in such a way that multiple process engine can run simultaneously and these engines have the same view of running process instance at one point of time. So, if a process instance is interrupted on one host then it can be continued

at another host. However implementing it has been a challenge. Our implementation aims to address this challenge.

To get an overall idea of how the system works, consider a number of hosts registered to a network and each of the host is defined by certain set of capabilities. Since the system is decentralized each host contains a copy of the migration logic, a process engine running as a web application inside a web server. All these web applications points towards one centralized database. Here, the communication with the database is not frequent and is only done when the check pointing has to be done during the execution of the process instance. We use the database to store and retrieve the constraint expressions and related task meta-information. When a process instance is started it starts execution of the workflow, as soon as an activity with a constraint defined on it is encountered the host invokes the migration logic on itself. This migration logic is responsible for evaluating the constraint based on its type and making the migration decision for the element.

For simplicity we describe a complete lifecycle of a process instance. A host machine can be registered with the system via a performer management interface. It is here that the host capabilities are specified. Then the designer creates the process definition by specifying the constraints on activities using BPMN artifacts. After the design phase, the process definition is parsed and deployed in the system as described in previous chapter. Now, the process is ready for instantiation. We describe the distributed execution of a process instance using a sequence diagram shown in Figure 7.3. A process instance is started at one host as soon as a constrained activity is encountered an execution listener is invoked which in return starts a constraint evaluation thread and gives the control back to the execution. The basic idea of giving back the control to the process instance execution, since the constrained activity is an Async task, it waits

for the external signal. The constraint evaluation thread obtains the meta-information about the process instance like process variables, job information, expression tree representation from the database. These information is then used the evaluate the constraint expression against the current host, if the constraint succeeds then the execution is continued on the same host and if the constraint fails then another host is selected by evaluating constraint with other hosts in the system and the

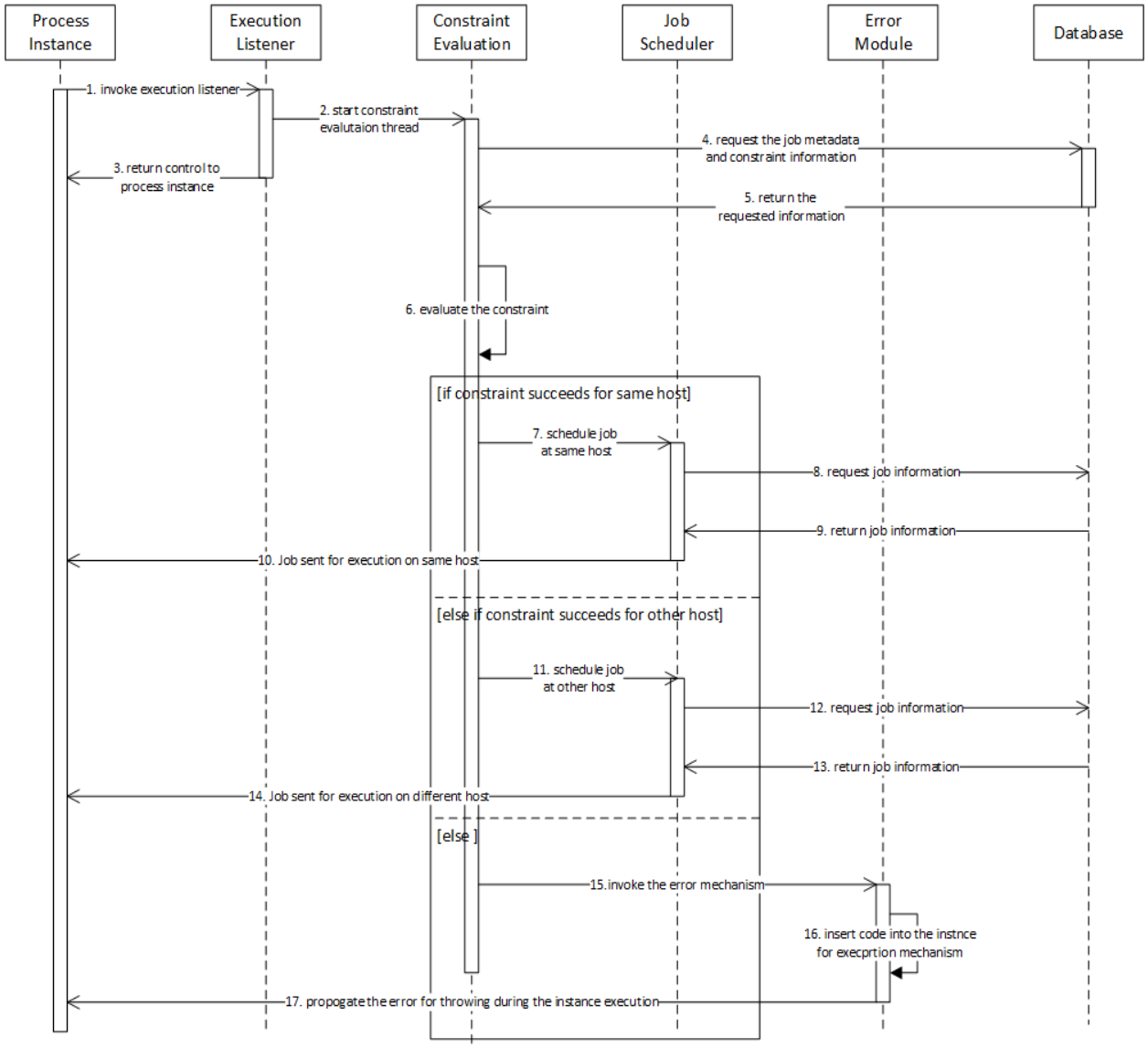


Figure 7.3: Sequence diagram for distributed process execution

execution is continued on that machine. It is possible that none of the host satisfies the constraint requirement. In this case the system throws an exception for the constraint and ends the instance execution. we allow the designer to decide the handling of these exceptions.

### 7.3 Tools and technologies

The whole system has been implemented using Java 1.7 [43]. All the components have been provided as deployable Web Applications which are run inside JBoss Application server [38]. We use MySql [40] database in the backend for Activiti. Also, JavaCC [36] has been used to implement the parser for validating the constraint expressions. There are a number of threads created at a single point of time in the application, hence we have two Java thread pools each for constraint evaluations and job executions for managing the number of threads. Activiti provides REST interface to manage the application hence we extend the functionality of this interface to provide additional functionality.

# CHAPTER 8

# EVALUATION

To evaluate our system we have developed a structure of the GlycoQuant IDAWG™ workflow in BPMN 2.0. To emulate the decentralized behavior we create a test bed comprising of three host machines, each having its own copy of the process engine and the migration logic running inside a JBoss container. These host machines are described using various performer capabilities such as the quality of service, input and output files, expensive license requirements among others. We ran three test cases to show different runtime behavior of the process.

- The test case shows the centralized execution of the system. Sometimes the system executing the workflow instance might be capable of completing all the tasks.
- The test case exhibits the process of migrating the workflow instances at runtime based on the constraint evaluations.
- This test case shows the handling of the errors in case of an exception.

GlycoQuant IDAWG™ BPMN Process

As described in the motivation of the thesis IDWAG™ is scientific workflow application used for performing quantitative analysis in glycomics. Figure 8.1 shows an emulation design of the IDAWG™ process enriched with CDL expression for runtime migration of a process instance. The description of the workflow is as follows.

1. A scientist starts a process instance by running the mass spectrometer experiment. The output of this experiment is large amount of data required to perform the quantitative analysis.

2. The data generated is then preprocessed to transform into an intermediate format required for the simulation step. The raw data generated from the spectrometer experiments may be very large and, to avoid large data transfer overhead, the preprocessing task can be executed at the host machine where the raw experimental data is generated.
3. The intermediate data then undergoes simulation and optimization followed by visualization. These tasks may sometimes have high resource requirements and prior tasks may not need much computation power, therefore to increase efficiency these computations can be moved different machines.

A typical process instance of IDAWG™ may run for many minutes, hence we focus on emulating the runtime behavior of the system rather than focusing on actual execution of the complete workflow.

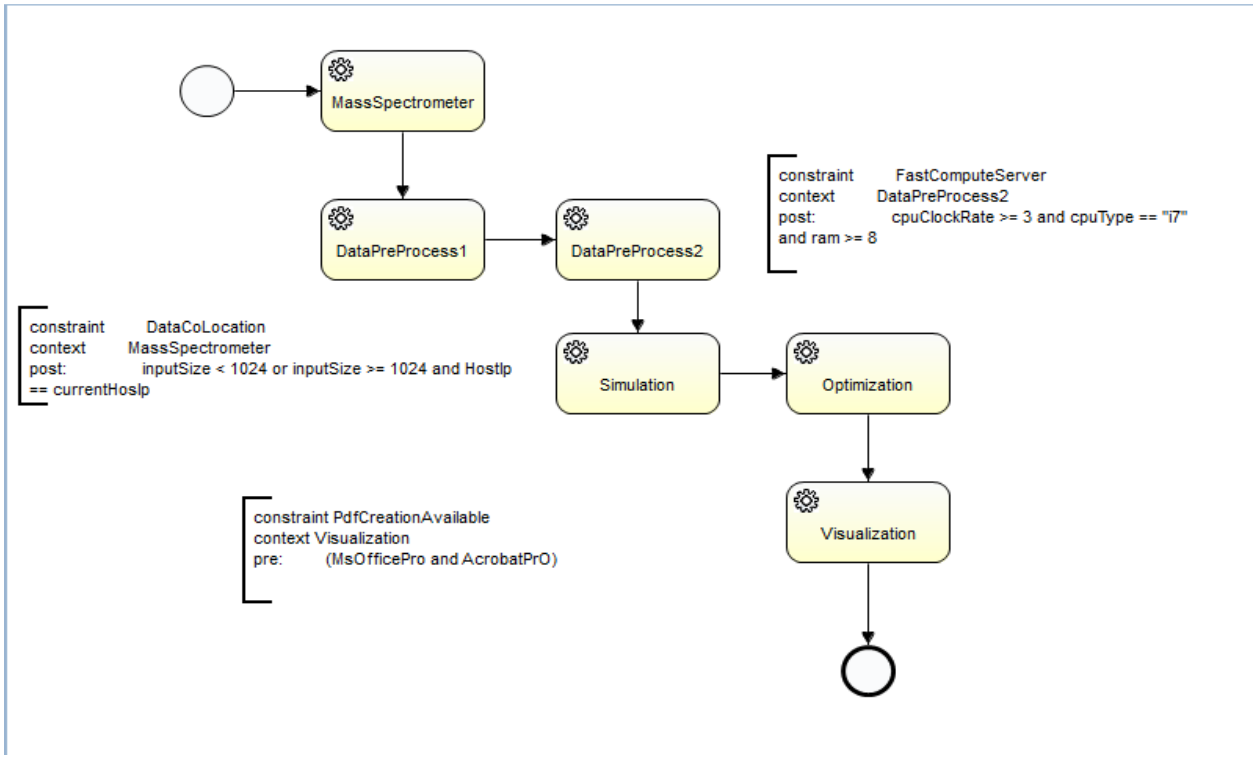


Figure 8.1: IDAWG™ with CDL expressions

Figure 8.2 shows the execution of IDAWG™ process instance on only one host machine. The constraints are evaluated at runtime such that all the tasks map to the same machine.

```
INFO   _____
INFO   ******************************************************
INFO   Simulating : MassSpectrometer Task
INFO   ******************************************************
INFO   Evaluating Post Condition on Execution : 135320..........
INFO   Constraint: DataCoLocation satisfied on same Host: 172.20.5.143
INFO   Sending job : 135324 to Host : 172.20.5.143
INFO   Job : 135324 Scheduled at Host : 172.20.5.143
INFO   _____
INFO   ******************************************************
INFO   Simulating : DataPreProcess1 Task
INFO   ******************************************************
INFO   _____
INFO   ******************************************************
INFO   Simulating : DataPreProcess2 Task
INFO   ******************************************************
INFO   Evaluating Post condition on Execution : 135320.........
INFO   Constraint: FastComputePower satisfied on same Host: 172.20.5.143
INFO   Sending job : 135327 to Host : 172.20.5.143
INFO   Job : 135327 Scheduled at Host : 172.20.5.143
INFO   _____
INFO   ******************************************************
INFO   Simulating : Simulation Task
INFO   ******************************************************
INFO   _____
INFO   ******************************************************
INFO   Simulating : Optimization Task
INFO   ******************************************************
INFO   Evaluating Pre Condition on Execution : 135320..........
INFO   Constraint: PdfCreationAvailable satisfied on same Host: 172.20.5.14
INFO   Job : 135330 to run on same Host: 172.20.5.143
INFO   Sending job : 135330 to Host : 172.20.5.143
INFO   Job : 135330 Scheduled at Host : 172.20.5.143
INFO   _____
INFO   ******************************************************
INFO   Simulating : Visualization Task
INFO   ******************************************************
```

Figure 8.2: Testcase1 server log host1

Figure 8.3, 8.4, 8.5 shows the server logs for host1, host 2, host 3 which exhibit the decentralized behavior of the process instance.

```
INFO   _____________________________________________________
INFO   *****************************************************
INFO   Simulating : MassSpectrometer Task
INFO   *****************************************************
INFO   Evaluating Post Condition on Execution : 135301.........
INFO   Constraint: DataCoLocation satisfied on same Host: 172.20.5.143
INFO   Job : 135305 to run on same Host: 172.20.5.143
INFO   Sending job : 135305 to Host : 172.20.5.143
INFO   Job : 135305 Scheduled at Host : 172.20.5.143
INFO   _____________________________________________________
INFO   *****************************************************
INFO   Simulating : DataPreProcess1 Task
INFO   *****************************************************
INFO   _____________________________________________________
INFO   *****************************************************
INFO   Simulating : DataPreProcess2 Task
INFO   *****************************************************
INFO   Evaluating Post condition on Execution : 135301.........
INFO   Constraint on same host failed..........
INFO   Constraint: FastComputeServer satisfied on Host: 128.192.62.248
INFO   Sending job : 135308 to Host : 128.192.62.248
INFO   Job : 135308 Scheduled at Host : 128.192.62.248
```

Figure 8.3: Testcase2 server log host1

```
INFO   _____________________________________________________
INFO   *****************************************************
INFO   Simulating : Simulation Task
INFO   *****************************************************
INFO   _____________________________________________________
INFO   *****************************************************
INFO   Simulating : Optimization Task
INFO   *****************************************************
INFO   Evaluating Pre Condition on Execution : 135301.........
INFO   Constraint on same host failed..........
INFO   Constraint: PdfCreationAvailable satisfied on Host: 128.192.62.243
INFO   Sending job : 135403 to Host : 128.192.62.243
INFO   Job : 135403 Scheduled at Host : 128.192.62.243
```

Figure 8.4: Testcase2 server log host2

```
INFO   _____________________________________________________
INFO   *****************************************************
INFO   Simulating : Visualization Task
INFO   *****************************************************
```

Figure 8.5: Testcase3 server log host 3

Exceptions in DIFLOW can be handled in different ways. But the underlying approach is to replace the executable code of the task by exception handling service at runtime. Once the error code is invoked, the process engine can handle the exception at instance level. One approach for the designer is to define an error event sub process such that when an exception is thrown it runs the sub process instead of the actual sequence.

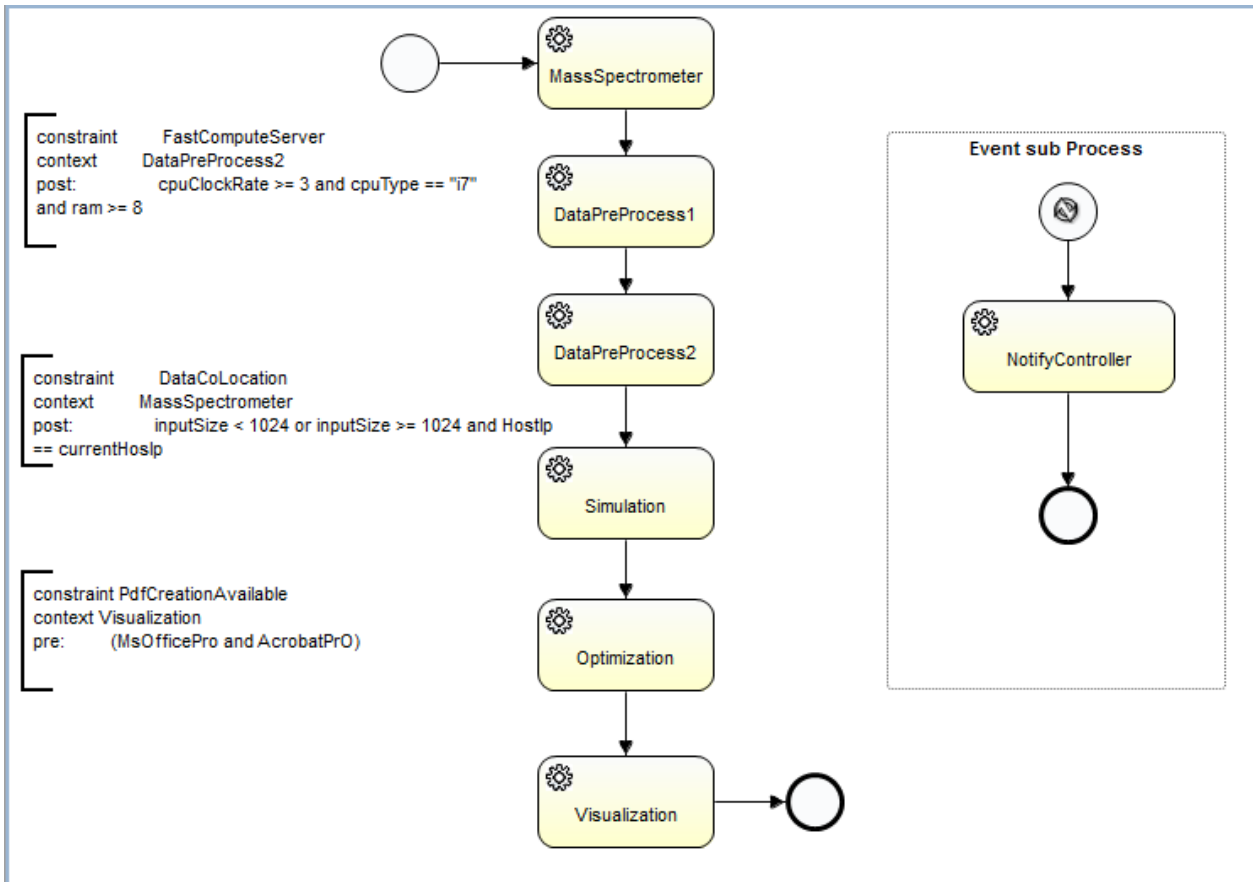


Figure 8.6: IDAWG™ with sub-process definition for exceptions

In Figure 8.6 we can see that if a constraint fails for all the machines then an exception is thrown which can be handled by the Event sub-process by catching the error code thrown by the task. If an error handling sub-process is not defined, then the instance is automatically ended when an exception occurs. Figure 8.7 shows the server log for handling the exception.

```
INFO    _____
INFO    ******************************************************
INFO    Simulating : MassSpectrometer Task
INFO    ******************************************************
INFO    Evaluating Post Condition on Execution : 135309.........
INFO    Constraint: DataCoLocation satisfied on same Host: 172.20.5.143
INFO    Job : 135313 to run on same Host: 172.20.5.143
INFO    Sending job : 135313 to Host : 172.20.5.143
INFO    Job : 135313 Scheduled at Host : 172.20.5.143
INFO    _____
INFO    ******************************************************
INFO    Simulating : DataPreProcess1 Task
INFO    ******************************************************
INFO    _____
INFO    ******************************************************
INFO    Simulating : DataPreProcess2 Task
INFO    ******************************************************
INFO    Evaluating Post condition on Execution : 135309.........
INFO    Constraint on same host failed.........
INFO    Constraint: FastComputeServer failed for all hosts
INFO    Evaluation of Constraint: FastComputePower for Task: DataPreProcess2
failed for all host..
INFO    Implementing exception mechanism for Task: DataPreProcess2
INFO    Sending job : 135316 to Host : 172.20.5.143
INFO    ******************************************************
INFO    An error occurred while evaluating the constraint
INFO    _____
INFO    ******************************************************
INFO    Simulating : NotifyController Task
INFO    ******************************************************
```

Figure 8.7: Testcase3 server log host1

Business Process BPMN

Business processes are often large and may span a number of departments within and across organizations. It is often necessary in business process outsourcing to execute the outsourced process at the third party site. Figure 8.8 shows an emulation of such type of process. The decisions made by exclusive-OR decides the path to follow in the process. The constraints described on the tasks decides the execution site of the host. Figure 8.9 shows the execution server log for the Testcase1 which runs using the first path as a result of OR gate and Figures 8.10 and 8.11 shows the execution server logs for Testcase2 which runs on the alternate path.

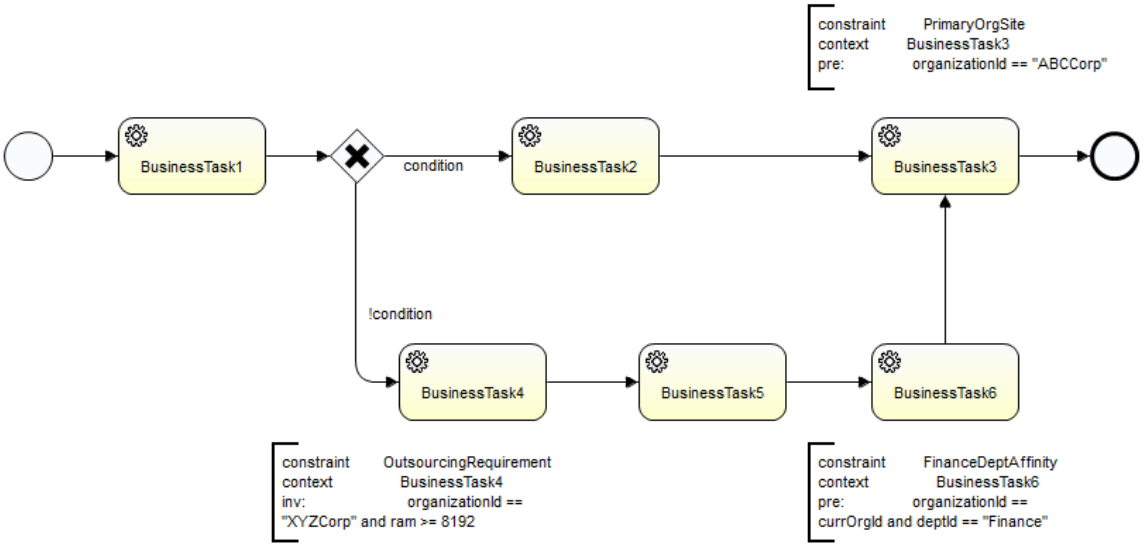


Figure 8.8: Business Process Example BPMN

```
INFO    _____________________________________________________
INFO    *****************************************************
INFO    Simulating : BusinessTask1 Task
INFO    *****************************************************
INFO    _____________________________________________________
INFO    *****************************************************
INFO    Simulating : BusinessTask2 Task
INFO    *****************************************************
INFO    Evaluating Pre condition on Execution : 135610.........
INFO    Constraint: PrimaryOrgSite satisfied on same Host: 172.20.5.143
INFO    Sending job : 135614 to Host : 172.20.5.143
INFO    Job : 135614 Scheduled at Host : 172.20.5.143
INFO    _____________________________________________________
INFO    *****************************************************
INFO    Simulating : BusinessTask3 Task
INFO    *****************************************************
```

Figure 8.9: Business Process Testcase1 server log host1

```
INFO   _____
INFO   ******************************************************
INFO   Simulating : BusinessTask1 Task
INFO   ******************************************************
INFO   Evaluating Pre Condition on Execution : 135713.........
INFO   Constraint on same host failed..........
INFO   Constraint: OutsourcingRequirement satisfied on Host: 128.192.62.248
INFO   Sending job : 135716 to Host : 128.192.62.248
INFO   Job : 135716 Scheduled at Host : 128.192.62.248
INFO   _____
INFO   ******************************************************
INFO   Simulating : BusinessTask3 Task
INFO   ******************************************************
```

Figure 8.10: Business Process Testcase2 server log host1

```
INFO   _____
INFO   ******************************************************
INFO   Simulating : BusinessTask4 Task
INFO   ******************************************************
INFO   Evaluating Post condition on Execution : 135713.........
INFO   Constraint: OutsourcingRequirement satisfied on same Host:
128.192.62.248
INFO   Sending job : 135719 to Host : 128.192.62.248
INFO   Job : 135719 Scheduled at Host : 128.192.62.248
INFO   _____
INFO   ******************************************************
INFO   Simulating : BusinessTask5 Task
INFO   ******************************************************
INFO   Evaluating Pre Condition on Execution : 135713.........
INFO   Constraint: FinanceDeptAffinity satisfied on same Host:
128.192.62.248
INFO   Sending job : 135721 to Host : 128.192.62.248
INFO   Job : 135721 Scheduled at Host : 128.192.62.248
INFO   _____
INFO   ******************************************************
INFO   Simulating : BusinessTask6 Task
INFO   ******************************************************
INFO   Evaluating Pre Condition on Execution : 135713.........
INFO   Constraint on same host failed..........
INFO   Constraint: PrimaryOrgSite satisfied on Host: 172.20.81.102
INFO   Sending job : 135722 to Host : 172.20.81.102
INFO   Job : 135722 Scheduled at Host : 172.20.81.102
```

Figure 8.11: Business Process Testcase2 server log host2

One of the main objectives of distributed workflow management system is to facilitate the parallel processing of the simultaneous tasks in the process. A business process may have a parallel fork (Exclusive-AND). DIFLOW offers such functionality using two approaches. The

first use case in Figure 8.12 shows a business process with and split with no constraints specified on the parallel tasks such that when DIFLOW constraint evaluator encounters the AND gateway result is always true and the job scheduler dynamically schedules the jobs on different hosts. Figures 8.13, 8.14 and 8.15 shows the server logs for the parallel processing of the process in Figure 8.12

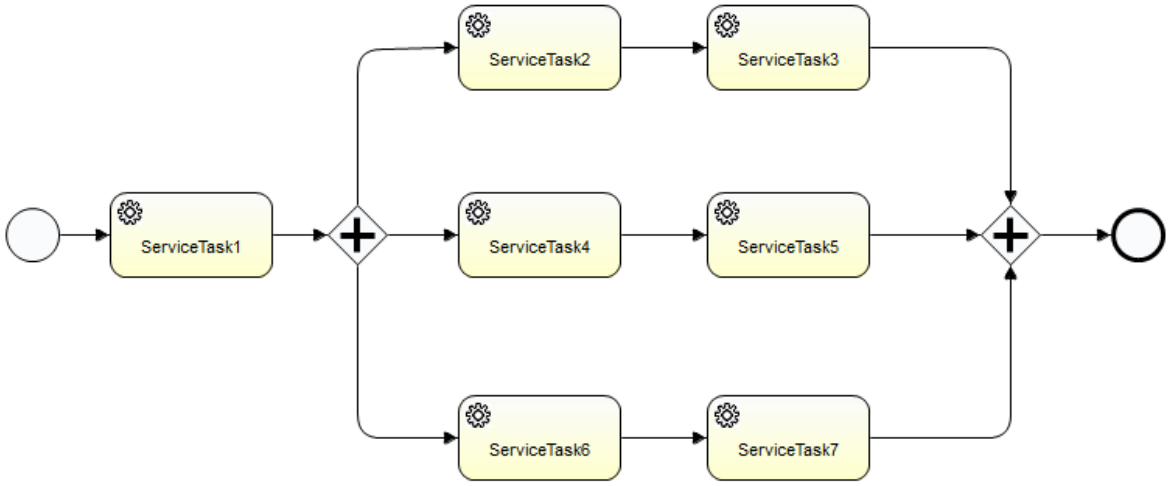


Figure 8.12: Business Process with parallel fork 1

```
INFO   ____________________________________________________
INFO   ****************************************************
INFO   Simulating : ServiceTask1 Task
INFO   ****************************************************
INFO   Sending job : 135324 to Host : 172.20.5.143
INFO   Job : 135324 Scheduled at Host : 172.20.5.143
INFO   ____________________________________________________
INFO   ****************************************************
INFO   Simulating : ServiceTask2 Task
INFO   ****************************************************
INFO   Sending job : 135325 to Host : 128.192.62.243
INFO   ____________________________________________________
INFO   ****************************************************
INFO   Simulating : ServiceTask3 Task
INFO   ****************************************************
INFO   Job : 135325 Scheduled at Host : 128.192.62.243
INFO   Sending job : 135326 to Host : 128.192.62.248
INFO   Job : 135326 Scheduled at Host : 128.192.62.248
```

Figure 8.13: Business Process with parallel fork 1 server log host1

```
INFO   _____________________________________________________
INFO   *****************************************************
INFO   Simulating : ServiceTask4 Task
INFO   *****************************************************
INFO   _____________________________________________________
INFO   *****************************************************
INFO   Simulating : ServiceTask5 Task
INFO   *****************************************************
```

Figure 8.14: Business Process with parallel fork 1 server log host2

```
INFO   _____________________________________________________
INFO   *****************************************************
INFO   Simulating : ServiceTask6 Task
INFO   *****************************************************
INFO   _____________________________________________________
INFO   *****************************************************
INFO   Simulating : ServiceTask7 Task
INFO   *****************************************************
```

Figure 8.15: Business Process with parallel fork 1 server log host3

Another approach for executing process forks is by specifying pre: condition constraints on the first tasks after an AND gateway, such that the designer provides the necessary requirements for the execution of the tasks. This approach is evaluated using example process definition shown in Figure 8.16. Similarly, Figures 8.17, 8.18 and 8.19 show the server logs for execution of process instance shown in Figure 8.16

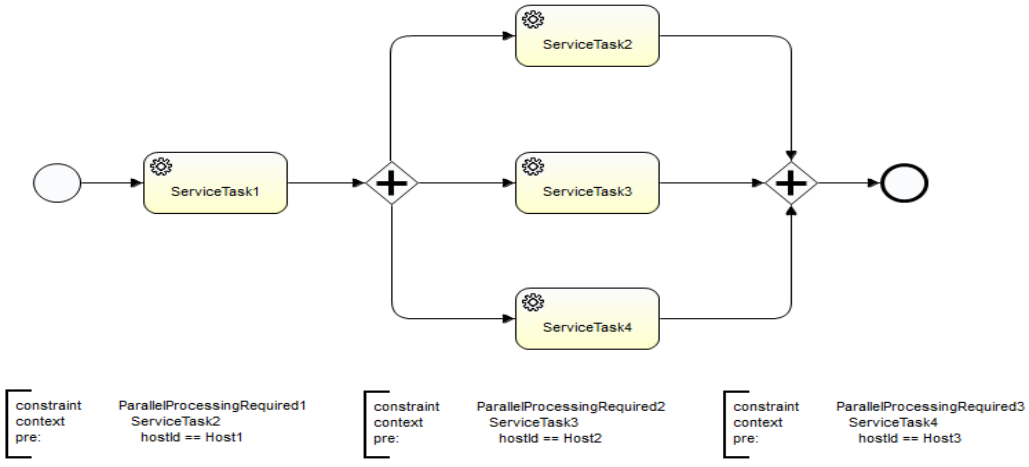


Figure 8.16: Business Process with parallel fork 2

```
INFO   _____________________________________________________
INFO   *****************************************************
INFO   Simulating : ServiceTask1 Task
INFO   *****************************************************
INFO   Evaluating pre Condition on Execution : 137220.........
INFO   Constraint: ParallelProcessingRequired1 satisfied on same Host:
172.20.5.143
INFO   Sending job : 137222 to Host : 172.20.5.143
INFO   Job : 137222 Scheduled at Host : 172.20.5.143
INFO   _____________________________________________________
INFO   *****************************************************
INFO   Simulating : ServiceTask2 Task
INFO   *****************************************************
INFO   Evaluating Pre Condition on Execution : 137220.........
INFO   Constraint on same host failed.........
INFO   Constraint: ParallelProcessingRequired2 on Host: 128.192.62.243
INFO   Sending job : 137223 to Host : 128.192.62.243
INFO   Job : 137223 Scheduled at Host : 128.192.62.243
INFO   Evaluating Pre Condition on Execution : 137220.........
INFO   Constraint on same host failed.........
INFO   Constraint: ParallelProcessingRequired3 on Host: 128.192.62.248
INFO   Sending job : 137224 to Host : 128.192.62.248
INFO   Job : 137224 Scheduled at Host : 128.192.62.248
```

Figure 8.17: Business Process with parallel fork 2 server log host1

```
INFO   _____________________________________________________
INFO   *****************************************************
INFO   Simulating : ServiceTask3 Task
INFO   *****************************************************
```

Figure 8.18: Business Process with parallel fork 2 server log host2

```
INFO   _____________________________________________________
INFO   *****************************************************
INFO   Simulating : ServiceTask4 Task
INFO   *****************************************************
```

Figure 8.19: Business Process with parallel fork 2 server log host3

# CHAPTER 9

# CONCLUSION AND FUTUREWORK

In this thesis, we have presented an approach for distributed workflow execution by integrating information into the process model. The approach uses the basics of object constraint language to fully facilitate the specification of necessary information in the process definition. The implementation of the project shows the viability of the concepts explained in the previous chapters. We presented a meta-model which are described using requirement specifications for integrating it into the process definition. These specifications are declared using Constraint Definition Language (CDL) comprising of context information and capability expressions. The constraints can be represented as pre-conditions, post-conditions and invariants that validate the executions of the tasks in a declarative way. Also, we have introduced an approach to enable the specifications of these constraints for the process definition using BPMN 2.0 constructs.

In the second part, we design and implement a system prototype for the runtime enactment of such process definition based on the decisions made by evaluating the information in a distributed execution environment.

We propose a few things in the future for further refining the system. A comparison of the centralized execution of Activiti process engine and decentralized execution of DIFLOW would be a good benchmark for an assessment. We would like to perform timing experiments on DIFLOW to measure the speedup acquired as a result of the execution of a distributed process definition. Currently, the system relies on the static values of performer capabilities fetched from the database. Our future plan would be to integrate DIFLOW with grid systems and cloud

technologies such that the performer capabilities that could be updated at run time. Also, the system focuses more on migrating the process instances when a set of service tasks are involved, we would like to further extend the functionality for different types of BPMN tasks like script tasks, mail tasks among others. We also would like to extend the BPMN specification by creating an artifact for the constraint declaration rather than using text annotation artifacts for this purpose. Though both artifacts would be similar, the distinction would provide a better abstraction in the process definition for the designer. The Web interface for performer management can also be integrated with the Activiti designer for simple usability.

## REFERENCES


[1] Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S. and others 2003. Business process execution language for web services. version.

[2] Baresi, L., Maurino, A. and Modafferi, S. 2007. Towards distributed bpel orchestrations. *Electronic Communications of the EASST*. 3, (2007).

[3] Brandic, I., Pllana, S. and Benkner, S. 2006. High-level composition of QoS-aware Grid workflows: an approach that considers location affinity. *Workflows in Support of Large-Scale Science, 2006. WORKS'06. Workshop on* (2006), 1–10.

[4] Buyya, R., Broberg, J. and Goscinski, A.M. 2010. *Cloud computing: Principles and paradigms*. John Wiley & Sons.

[5] Deelman, E., Singh, G., Su, M.-H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G.B., Good, J. and others 2005. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*. 13, 3 (2005), 219–237.

[6] Dogac, A., Gokkoca, E., Arpinar, S., Koksal, P., Cingil, I., Arpinar, B., Tatbul, N., Karagoz, P., Halici, U. and Altinel, M. 1998. Design and implementation of a distributed workflow management system: Metuflow. *Workflow Management Systems and Interoperability*. Springer. 61–91.

[7] Frey, J. 2002. Condor DAGMan: Handling inter-job dependencies.

[8] Frey, J., Tannenbaum, T., Livny, M., Foster, I. and Tuecke, S. 2002. Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing*. 5, 3 (2002), 237–246.

[9] Gerhards, M., Sander, V. and Belloum, A. 2012. About the flexible Migration of Workflow Tasks to Clouds. *CLOUD COMPUTING 2012, The Third International Conference on Cloud Computing, GRIDs, and Virtualization* (2012), 82–87.

[10] Hollingsworth, D. 1995. WfMC , The workflow reference model. (1995).

[11] Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M.R., Li, P. and Oinn, T. 2006. Taverna: a tool for building and running workflows of services. *Nucleic acids research*. 34, suppl 2 (2006), W729–W732.

[12] Khalaf, R., Kopp, O. and Leymann, F. 2008. Maintaining data dependencies across bpel process fragments. *International Journal of Cooperative Information Systems*. 17, 03 (2008), 259–282.

[13] Khalaf, R. and Leymann, F. 2012. Coordination for fragmented loops and scopes in a distributed business process. *Information Systems*. 37, 6 (2012), 593–610.

[14] Khalaf, R. and Leymann, F. 2006. E role-based decomposition of business processes using bpel. *Web Services, 2006. ICWS'06. International Conference on* (2006), 770–780.

[15] Kochut, K., Arnold, J., Sheth, A., Miller, J., Kraemer, E., Arpinar, B. and Cardoso, J. 2003. IntelliGEN: A distributed workflow system for discovering protein-protein interactions. *Distributed and Parallel Databases*. 13, 1 (2003), 43–72.

[16] Kochut, K.J., Sheth, A.P. and Miller, J.A. 1998. ORBWork: A CORBA-based fully distributed, scalable and dynamic workflow enactment service for METEOR. *Large Scale Distributed Information Systems Lab, Department of Computer Science, University of Georgia, Athens, GA*. (1998).

[17] Leymann, F. 2011. BPEL vs. BPMN 2.0: Should you care? *Business Process Modeling Notation*. Springer. 8–13.

[18] Liu, S., Correa, M. and Kochut, K. 2013. An Ontology-Aided Process Constraint Modeling Framework for Workflow Systems. *eKNOW 2013, The Fifth International Conference on Information, Process, and Knowledge Management* (2013), 178–183.

[19] Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E.A., Tao, J. and Zhao, Y. 2006. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience*. 18, 10 (2006), 1039–1065.

[20] Martin, D., Wutke, D. and Leymann, F. 2008. A novel approach to decentralized workflow enactment. *Enterprise Distributed Object Computing Conference, 2008. EDOC'08. 12th International IEEE* (2008), 127–136.

[21] Model, B.P. Notation (BPMN), v. 2.0, 2011. *OMG: www. omg. org/spec/BPMN/2.0*.

[22] Murata, T. 1989. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*. 77, 4 (1989), 541–580.

[23] Papazoglu, M. and Schlageter, G. 1997. *Cooperative information systems: trends and directions*. Academic Press.

[24] Pesic, M., Schonenberg, M., Sidorova, N. and Aalst, W.M. van der 2007. Constraint-based workflow models: Change made easy. *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*. Springer. 77–94.

[25] Rademakers, T. 2012. *Activiti in Action: Executable business processes in BPMN 2.0*. Manning Publications Co.

[26] Sheth, A., Worah, D., Kochut, K.J., Miller, J.A., Zheng, K., Palaniswami, D. and Das, S. 1997. The METEOR workflow management system and its use in prototyping significant healthcare applications. *Proc. of the Toward an Electronic Patient Record Conf.(TEPR'97)* (1997), 267–278.

[27] Specification, O.I. 2006. Object Management Group. *Needham, MA, USA*. 2, 2 (2006).

[28] Sun, S.X., Zeng, Q. and Wang, H. 2011. Process-mining-based workflow model fragmentation for distributed execution. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*. 41, 2 (2011), 294–310.

[29] WfMC, G. 1999. Terminology and Glossary. Document No WFMC-TC-1011. Workflow Management Coalition. Winchester.

[30] Yu, J. and Buyya, R. 2005. A taxonomy of scientific workflow systems for grid computing. *ACM Sigmod Record*. 34, 3 (2005), 44–49.

[31] Zaplata, S., Hamann, K., Kottke, K. and Lamersdorf, W. 2010. Flexible execution of distributed business processes based on process instance migration. *Journal of Systems Integration*. 1, 3 (2010), 3–16.

[32] Activiti BPM. available from http://activiti.org.

[33] 2009. Business Process Management (BPM) Center of Excellence (CoE) Glossary. Available from https://www.ftb.ca.gov/aboutftb/projects/itsp/bpm_glossary.pdf.

[34] Eclipse IDE. Available from: https://www.eclipse.org.

[35] http://en.wikipedia.org/wiki/Object_Constraint_Language.

[36] Java Compiler Compiler (JAVACC) . Available from: https://javacc.java.net/.

[37] JBoss jBPM. Available from https://www.jboss.org/jbpm.

[38] JBoss. Available from: http://www.jboss.org.

[39] KIS-BPM. Availabe from http://kisbpm.com.

[40] MySql. Available at: http://www.mysql.com/.

[41] Object Constraint Language. Available at http://www.omg.org/spec/OCL/2.4/.

[42] Object-capability Model at http://en.wikipedia.org/wiki/Object-capability_model.

[43] Oracle. Java Platform, Standard Edition 7: API Specification. 2011; Available from: http://download.oracle.com/javase/7/docs/api/.