Process Definition, Verification and Simulation
—An Implementation in a Visual Modeling Tool for a Workflow Management System

by

Hong Zhang

A dissertation submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

(Management Information System)

Under the supervision of Professor Ludwig Nastansky

at the

University of Paderborn
Germany

2000
ACKNOWLEDGEMENTS

This work is the result of 5.5 years of research, development and application of process modeling and simulation for a workflow management system (the Espresso WfMS) during the author’s affiliation with GCC (Groupware Competence Center) in the University of Paderborn and the software company of PAVONE AG.

I am very grateful to Mr. Professor Ludwig Nastansky, my mentor and principal supervisor, who helped the dissertation in many different ways, particularly who offered me a job at the university since September 1994 so that I could work for PAVONE to gather necessary experience and to implement the theory and methods discussed in the dissertation in the product of the ProcessModeler. I also wish to express my gratitude to Mr. Professor XX, the second supervisor of the dissertation, and other members of the promotion commission, Professor XXX and Professor XXX.

I am indebted to Mr. Howard Almond, who revised the whole work and provided me with a lot of suggestions and diverse help during my doctor work, and Mr. Rolf Kremer, who kindly read a preliminary draft and provided me with his critical comments and various advises. Mr. Stefan Meyer also deserves my gratitude for his generous help in correcting and reviewing the work.

I would like to thank Mr. Francis Sarver for implementing the workflow control data discussed in this work in the workflow engine of the Espresso WfMS and Mr. Jens Winkelmann for his ideas and urging me on to finish the dissertation. My thanks go also to Mr. Ulrich Foerster for his efforts in enabling me to retain my job at the university.

Appreciation is due to Ms. Amy Jean of CFT and other colleagues in PAVONE and its partner firms for offering me a lot of practical suggestions for improving the product PM by using it in the real world for the design and simulation of business processes.

I would like to pay a tribute to the original GroupFlow team of PAVONE. It was the product devised by this team with which I began my doctor work.

Finally, I wish to thank my mother, who worried for me a lot during my doctor work and never forgot to bless me for a happy, successful life.

Hong Zhang
Paderborn, May 2000
CONTENTS

Acknowledgements .............................................................. iii
Preface .............................................................................. xi
  Motivation ........................................................................ xi
  Objective .......................................................................... xii
  Outline ............................................................................ xiv
Part One: Process Modeling and Verification 
  1 Introduction of Workflow management system ................. 1
    1.1 Basic Workflow Terminology .................................... 1
    1.2 The Espresso Workflow Management System ............... 4
      1.2.1 The ProcessModeler .......................................... 5
      1.2.2 Application Database ....................................... 7
      1.2.3 Organizational Model ....................................... 8
    1.3 Process Examples .................................................... 9
    1.4 Conclusion ............................................................ 11
  2 Process Definition ....................................................... 13
    2.1 Activity and Link .................................................... 13
    2.2 Start Activity ....................................................... 14
    2.3 Algorithms for Keeping a Process Definition ................ 16
    2.4 Conclusion ............................................................ 17
  3 Routing Options .......................................................... 19
    3.1 Routing Option Definition ...................................... 19
    3.2 Algorithm for Getting Invalid IT Resource Activities .... 22
    3.3 End Activity ......................................................... 22
    3.4 Conclusion ............................................................ 24
  4 Path and Cycle ............................................................ 25
    4.1 Paths ................................................................. 25
      4.1.1 Elementary Path .............................................. 27
      4.1.2 Algorithm for Finding a Path* ......................... 28
    4.2 Reachability and Criticality ..................................... 30
      4.2.1 Reachability .................................................. 30
      4.2.2 Criticality ...................................................... 31
      4.2.3 Algorithm for Determining Reachable and Critical ... 32
    4.3 Cycles ................................................................. 35
      4.3.1 Algorithm for Determining Activity Reachable to End .. 36
      4.3.2 Algorithm for Getting Elementary Cycles ............. 39
      4.3.3 Algorithm for Getting Infinite Elementary Cycles ... 40
    4.4 Algorithm for Determining Potential Start Activities ...... 42
    4.5 Conclusion ............................................................ 44
  5 Split and Join ............................................................. 46
    5.1 Split Activity ........................................................ 46
      5.1.1 Algorithm for Updating Split Activities ................ 47
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1.2</td>
<td>Ancestor of a Split Activity</td>
<td>48</td>
</tr>
<tr>
<td>5.1.3</td>
<td>Algorithm for Getting Ancestors of a Split Activity</td>
<td>49</td>
</tr>
<tr>
<td>5.2</td>
<td>Join Activity</td>
<td>50</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Examples of Join Activities</td>
<td>58</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Algorithm for Getting Ancestor Joins</td>
<td>57</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Algorithm for Determining Joins</td>
<td>58</td>
</tr>
<tr>
<td>5.3</td>
<td>Parallel Activity</td>
<td>59</td>
</tr>
<tr>
<td>5.4</td>
<td>Workflow Control Data</td>
<td>62</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Algorithm for Getting the First Split Activity</td>
<td>62</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Algorithm for Determining Workflow Control Data</td>
<td>63</td>
</tr>
<tr>
<td>5.5</td>
<td>Conclusion</td>
<td>66</td>
</tr>
<tr>
<td>6</td>
<td>Deadlock</td>
<td>68</td>
</tr>
<tr>
<td>6.1</td>
<td>Deadlock Opportunity</td>
<td>68</td>
</tr>
<tr>
<td>6.2</td>
<td>Join Priority and Deadlock Release</td>
<td>69</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Algorithm for Getting Elementary Deadlocks</td>
<td>70</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Algorithm for Grouping Deadlocks</td>
<td>72</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Example: Deadlock Release</td>
<td>74</td>
</tr>
<tr>
<td>6.2.3.1</td>
<td>Join Priority Order 1</td>
<td>75</td>
</tr>
<tr>
<td>6.2.3.2</td>
<td>Join Priority Order 2</td>
<td>75</td>
</tr>
<tr>
<td>6.2.3.3</td>
<td>Join Priority Order 3</td>
<td>76</td>
</tr>
<tr>
<td>6.2.3.4</td>
<td>Join Priority Order 4</td>
<td>77</td>
</tr>
<tr>
<td>6.2.3.5</td>
<td>Join Priority Order 5</td>
<td>78</td>
</tr>
<tr>
<td>6.2.3.6</td>
<td>Join Priority Order 6</td>
<td>79</td>
</tr>
<tr>
<td>6.3</td>
<td>Algorithms for Handling Deadlocks</td>
<td>79</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Algorithm for Getting One Deadlock</td>
<td>79</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Algorithm for Transmitting Work to Activity</td>
<td>82</td>
</tr>
<tr>
<td>6.3.3</td>
<td>Algorithm for Getting Parallel Waiting Work Items</td>
<td>82</td>
</tr>
<tr>
<td>6.3.4</td>
<td>Algorithm for Releasing a Deadlock</td>
<td>83</td>
</tr>
<tr>
<td>6.3.5</td>
<td>Algorithm for Detecting Deadlocks</td>
<td>83</td>
</tr>
<tr>
<td>6.4</td>
<td>Conclusion</td>
<td>84</td>
</tr>
<tr>
<td>7</td>
<td>Summary</td>
<td>86</td>
</tr>
<tr>
<td>7.1</td>
<td>Definitions and Algorithms</td>
<td>86</td>
</tr>
<tr>
<td>7.2</td>
<td>Constraints in Process Modeling and Enactment</td>
<td>87</td>
</tr>
<tr>
<td>7.3</td>
<td>Further Work*</td>
<td>87</td>
</tr>
<tr>
<td>Part Two: Basic Simulation Knowledge</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Introduction of Simulation</td>
<td>91</td>
</tr>
<tr>
<td>8.1</td>
<td>Simulation</td>
<td>91</td>
</tr>
<tr>
<td>8.1.1</td>
<td>Disadvantages of Simulation</td>
<td>91</td>
</tr>
<tr>
<td>8.1.2</td>
<td>Why Simulation</td>
<td>92</td>
</tr>
<tr>
<td>8.2</td>
<td>Basic Simulation Terminology</td>
<td>94</td>
</tr>
<tr>
<td>8.2.1</td>
<td>System Categorization</td>
<td>94</td>
</tr>
<tr>
<td>8.2.2</td>
<td>Simulation Model</td>
<td>95</td>
</tr>
<tr>
<td>8.2.2.1</td>
<td>Next-Event Approach</td>
<td>96</td>
</tr>
<tr>
<td>8.2.2.2</td>
<td>Algorithm of Next-Event Approach</td>
<td>98</td>
</tr>
</tbody>
</table>
8.2.3 Variables ................................................................. 98  
  8.2.3.1 Decision Variable ........................................... 99  
  8.2.3.2 System Parameter ........................................... 100  
  8.2.3.3 State Variable ............................................... 100  
  8.2.3.4 Statistic Variable ......................................... 101  
  8.2.3.5 System Performance Criterion .......................... 103
8.3 Conclusion ....................................................................... 103
9 Statistic Theory and Methods ........................................... 105  
  9.1 Random Variable ..................................................... 105  
  9.2 Distribution Functions .......................................... 106  
    9.2.1 Uniform Distribution ....................................... 106  
    9.2.2 Normal Distribution ........................................ 108  
    9.2.3 Exponential Distribution ................................... 109  
    9.2.4 Gamma Distribution ........................................ 110  
    9.2.4.1 Algorithm for Drawing Gamma Distribution . . . 112  
    9.2.5 Empirical Distribution ...................................... 112  
    9.2.6 Student’s t-Distribution* .................................... 114  
    9.2.7 $\chi^2$ Distribution* ......................................... 115  
    9.2.8 F-Distribution* .................................................. 116  
  9.3 Estimation and Hypothesis .......................................... 117  
    9.3.1 Mean and Standard Deviation ............................. 117  
    9.3.2 Distribution Function ....................................... 118  
  9.4 Tests of Hypotheses* .............................................. 119  
    9.4.1 Introduction .................................................... 119  
    9.4.2 $t$-Test ............................................................ 121  
    9.4.3 $N(0, 1)$ Test ................................................... 122  
    9.4.4 $\chi^2$ Test ...................................................... 122  
    9.4.5 $F$-Test ........................................................... 124  
  9.5 An Example of Estimation and Test* ........................... 124  
  9.6 Conclusion ............................................................. 128
10 Random Variate Generation .......................................... 130  
  10.1 Introduction ........................................................ 130  
  10.2 Generating Random Number .................................... 131  
    10.2.1 The Power Residue Method ............................... 131  
    10.2.2 Algorithm for Generating Random Numbers ........... 133  
    10.2.3 Generating Random Numbers in Basic .................. 134  
  10.3 Testing and Validating Random Numbers ...................... 136  
    10.3.1 Frequency Test ............................................... 136  
    10.3.2 Increasing and Decreasing Runs ......................... 138  
    10.3.3 Other Tests .................................................... 140  
  10.4 Generating Random Variates .................................... 140  
    10.4.1 General Methods for Generating a Variate .......... 141  
      10.4.1.1 The Inverse Transformation Method ................ 141  
      10.4.1.2 The Rejection Method* ................................ 143  
    10.4.2 Uniformly Distributed Random Variates ............... 144
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.4.3 Normally Distributed Random Variates</td>
<td>145</td>
</tr>
<tr>
<td>10.4.3.1 Algorithm for Generating a Normal Variate</td>
<td>145</td>
</tr>
<tr>
<td>10.4.4 Exponentially Distributed Random Variates</td>
<td>146</td>
</tr>
<tr>
<td>10.4.4.1 Algorithm for Generating an Exponential Variate</td>
<td>146</td>
</tr>
<tr>
<td>10.4.4.2 Exponential Variates Greater than Zero*</td>
<td>147</td>
</tr>
<tr>
<td>10.4.5 Gamma Distributed Random Variates</td>
<td>147</td>
</tr>
<tr>
<td>10.4.5.1 Algorithm for Generating a Gamma Variate</td>
<td>147</td>
</tr>
<tr>
<td>10.4.5.2 Algorithm for Generating a Gamma Variate with Non-Integer Valued Shape Parameter*</td>
<td>148</td>
</tr>
<tr>
<td>10.4.6 Empirical Distributed Random Variates</td>
<td>149</td>
</tr>
<tr>
<td>10.4.6.1 Algorithm for Generating an Empirical Variate</td>
<td>150</td>
</tr>
<tr>
<td>10.4.7 The $\chi^2$, $t$- and $F$- Distributions*</td>
<td>151</td>
</tr>
<tr>
<td>10.5 Conclusion</td>
<td>151</td>
</tr>
</tbody>
</table>

**Part Three: WfMS Simulation**

11 Simulating Business Processes | 153
---|---
| 11.1 Process Settings | 153
| 11.1.1 Process Life Period Settings | 153
| 11.1.2 Process Creation Settings | 154
| 11.1.3 Routing Probability Settings | 157
| 11.1.4 Variable Settings | 158
| 11.1.5 Process Stop Settings | 161
| 11.2 Graphic Process States | 161
| 11.3 Data Structure of Simulated Work Items | 163
| 11.3.1 Work Record | 163
| 11.3.2 Work Protocol | 165
| 11.3.3 Algorithms Relevant to Simulated Work Items | 166
| 11.3.3.1 Algorithm for Connecting a Parallel Work | 166
| 11.3.3.2 Algorithm for Disconnecting a Parallel Work | 167
| 11.3.3.3 Algorithm for Copying a Work Protocol | 168
| 11.3.3.4 Algorithm for Eliminating Protocol Records | 168
| 11.3.3.5 Algorithm for Checking an Activity Split in a Protocol | 169
| 11.3.3.6 Algorithm for Getting All Split Activities in Protocols | 170
| 11.3.3.7 Algorithm for Eliminating a Work Record | 171
| 11.3.3.8 Algorithm for Getting Process Critical Paths | 173
| 11.4 Conclusion | 174

12 Simulating Resources | 176
---|---
| 12.1 Resource Specification | 176
| 12.1.1 Workflow Participant Specification | 177
| 12.1.1.1 Simulating Editor “Computed” | 177
14 General Simulation Phases ................................................................. 237
   14.1 History Data Collection ............................................................... 237
   14.2 Distribution Function Selection .................................................... 239
   14.3 Validation of the Simulation Model* ............................................. 241
      14.3.1 Determination of Expected System Performances ................. 241
      14.3.2 Some Theoretical Results for the Single-channel Single-station Queue ................................................................. 243
      14.3.2.1 The M/M/1 Queue .............................................................. 243
      14.3.2.2 The M/G/1 Queue .............................................................. 244
   14.4 Evaluation of Simulation Results* .............................................. 246
      14.4.1 Determination of Sample Size .............................................. 248
      14.4.2 Blocking ............................................................................. 249
   14.5 Conclusion ................................................................................. 253

References ............................................................................................ 255
Appendices ............................................................................................ 259
   Symbols in Definitions and Algorithms .............................................. 259
   Table 1. Selected Values of the \( t \)-Distribution ........................................ 260
   Table 2. Selected Values of the \( N(0, 1) \) Distribution .............................. 261
   Table 3. Selected Values of the \( \chi^2 \) Distribution ............................... 262
   Table 4. Selected Values of the \( F \)-Distribution for \( \alpha = 0.05 \) .............. 263
Indices .................................................................................................... 264
   Symbolic Definitions .......................................................................... 264
   Concept Definitions .......................................................................... 266
   Assumptions ...................................................................................... 268
   Algorithms ......................................................................................... 269
   Figures .............................................................................................. 270
   Examples .......................................................................................... 273

*: not implemented in the Espresso ProcessModeler
Motivation

“Since the early 1950’s there has developed an overwhelming interest in using digital computers to assist man’s quest for a better life and to increase his current productivity. The fact that digital computers have had a profound impact upon government, technology, business, and education need not be emphasized here” [Rosko, 1972, p. iii]

Comparing with the high efficiencies of automated manufacturing processes in industry, the efficiencies of management processes and business processes involved with people are quite low. “The increased wealth of the industrialized world during the twentieth century is due to ever-improving productivity in the manufacturing sector. This process has now reached the point where further enhancements to manufacturing processes are becoming less significant. To continue economic growth, it is now the turn of the service sectors to improve their efficiencies. In contrast to manufacturing, productivity in offices has barely changed in recent years, despite the widespread introduction of computers.” [Lawrence, 1997, p. 27]

“Workflow bridges the enterprise, from manufacturing to the office, from technology to organizational culture. It is this unifying force that ultimately binds an organization, its people and processes together. In this sense, workflow has always existed in all organizations, whether it is automated or not, the flow of material, information, and knowledge must be orchestrated in order to deliver a product or service. Because there is no such thing as a single step process, workflow is always present, in some fashion to manage the pieces from step to step. But this simple task, of managing the flow of work, is perhaps the single most important element of competitive advantage in mature markets, which have reached a stage of product, service, and positioning stability. At this point, competitive disparity can often only be diminished through quantum improvements in the redesign of underlying business processes. Add to this the global economic and competitive force in today’s business climate, and the automation of workflow becomes an imperative for survival.” [Koulopoulos, 1995, p. xv]

The tremendous development of information technology, especially the appearance and enhancement of groupware techniques, make it possible to improve the efficiency of business processes. Groupware “refers to a set of technologies that can be applied to improve the productivity of people working together in groups.” [Currid, 1994, p. 156]

Various workflow management systems (WfMS) developed in the 1990s help enterprises to realize the automation of worldwide business processes. “A workflow engine distributes, routes, and tracks documents according to a process defined in your application. Workflow enables you to coordinate and
streamline critical business activities across your organization, and with customers, partners, and suppliers.” [Toulemonde, 1998, p. 22]

Most organizations using WfMS are motivated by three factors (see [Lawrence, 1997, p. 6]):

- improved efficiency, leading to lower costs or higher workload capacity;
- improved control, resulting from standardization of procedures; and
- improved ability to manage processes, for performance problems are made explicit and understood.

WfMS are being used by many different types of organizations, in many different ways. For example (see [Lawrence, 1997, p. 7]):

- for insurance companies to speed up claims management while maintaining control over it;
- for government departments to improve efficiency in making decisions about paying social security benefits;
- for organizations of all types to improve the effectiveness of their customer service operations and order processing;
- to support routine internal administrative processes, such as personnel reporting and expense-claims management;
- to enable people to construct their own, customized, workflow processes to deal with their own specialized process responsibilities;
- to support even very complex processes, such as extremely large software development projects; etc.

WfMS can even be used by a virtual organization. The term Virtual Organization “is applied to a temporary coalition of several, legally independent organizations, with the purpose of offering a jointly manufactured product or jointly provide service to a customer who perceives the virtual organization as a singular entity.” [Riempp, 1998, p. 38] “Today there is no longer any question that widely dispersed office workers need efficient technical support by telecommunication and computer technology in order to meet the challenges of fast and flexible performance.” [Riempp, 1998, p. 23]

**Objective**

The prime objective of this work is to establish a methodology useful for process modeling and verification and able to be implemented in a practical visual modeling tool—ProcessModeler (PM). The pre-defined processes can then be simulated graphically with the simulator integrated in PM.

“When we try to solve a problem, we often draw a graph. A graph is often the simplest and easiest way to describe a system, a structure, or a situation.”
[Hu, 1982, p. 1] With PM, a process definition, which is a network of activities and connected by links, can be simply modeled as a graph. The automated business processes can then be created at one of the specified start activities of the process definition and be routed along the links from activity to activity. The diverse routing options of a links let the activity execution thread of a certain business process be dynamically determined, as the process behaves in the real world.

The activity network of a process definition can be flexibly designed with PM and can be very complex. However all the activities within a process definition should be reachable from a start activity. No infinite cycle in a process definition is allowed. The end activities where a business process can terminate are determined according to the activity network and routing options of links. PM determines for the workflow engine the split and join activities of a process definition—at a split activity a work item associated with a business process may be copied to several parallel work items; at a join activity parallel work items associated with the same business process must be merged into one work item. So unbalanced parallel activity execution threads in a process definition can prolong the duration of a business process. The workflow control data generated according to the algorithms discussed in the work is utilized by the workflow engine to determine whether a work item associated with a join activity should wait for joining. Waiting for joining may cause deadlock of a business process. PM can detect the deadlock opportunities of a process definition and let the designer to specify join priority for the workflow engine to release a deadlock.

An activity within a process definition is mostly assigned to an organizational role, which will be resolved to one or several people by the workflow engine at run-time for executing the work items associated with the activity. For execution of an activity, some materials may be used. Shortage of human and synchronous material resources can also lengthen the duration of a business process. Long duration of a business process means low service level. But superfluous resources engaged in an organization increase the resource costs and results in bad system efficiency.

“A workflow process definition which contains errors may lead to angry customers, back-log, damage claims, and loss of goodwill. Flaws in the design of a workflow definition may also lead to high throughput times, low service levels, and a need for excess capacity. This is why it is important to analyse a workflow process definition before it is put into production.” [Van der Aalst, 1998, p. 17]

Simulation is to make experiments on a model (or a simulator) which can sufficiently represent cause-and-effect relationships of a real world system. The simulation reports offer insight to the workloads, bottlenecks, resource allocation, throughput, productivity, and overall business cycle. By analyzing these, immediate decisions can be made to alter a process definition by reallocating resources, changing activity network, eliminating redundancy, or altering priorities of work. See [Lawrence, 1997, p. 37]
A WfMS is stochastic and dynamic. The states of the system vary with the occurrences of events and the time period between occurrences of two events (for example, the event that a person starts executing an activity and the event that he completes the activity) are not fixed. Before a simulation study, the analyst should be familiar with the feature of a distribution function so that they are correctly selected for generating a time period and other random data. The algorithm for generating a variate governed by a distribution function affects the validity of a simulation model.

To simulate a WfMS, a broad body of input data should be estimated upon the collected data or guessed by specialist so that they represent the features of the real world system as well as possible. The validity of the results gained from a simulation study is influenced by such factors as the techniques used in the collection of data and the analysis methods used in summarizing the data. Further, prior to its use, the simulator must be validated, or shown to actually represent the system being studied. In this work, the theory and methods requisite for the proper development and operation of the simulator will be presented.

Algorithms for process modeling and especially simulation are built upon some assumptions. The assumptions specify prerequisites, constraints, or principles for process designing and simulating. They should not diverge from the behavior of the real world business processes.

The algorithms included in this work are described for a general-purpose programming language, such as C++ and Visual Basic. They provide deep insights into the actual logical intricacies of PM. The techniques and algorithms discussed in the sections marked with “*” in this work are not implemented in PM.

Outline

The work is divided into three major parts. Part One delineates the theory and methods as well as the complete algorithms about process modeling and verification. The first chapter introduces the fundamental workflow concepts and the Espresso WfMS where the process definitions designed with PM are enacted. Chapter 2 to Chapter 6 concentrate on the symbolic definition and techniques of process modeling and verification. Complete algorithms for process definition and verification are discussed in this part. Before reading Chapter 2, the reader is recommended to read the first appendix for the descriptions of symbols used in definitions and algorithms.

Part Two covers the basic simulation knowledge and methods that is needed in simulations. The goals of the part are

(1) to outline the basic concepts of the simulation and the simulation model (Chapter 8);
(2) to review statistic theory and methods used in the simulation study for estimating distribution function of a random variable and testing the hypotheses (Chapter 9); and

(3) to provide the commonly used techniques for generating random variates governed by various distribution functions (Chapter 10).

Those readers with a sound background in these concepts and techniques can exclude this part, or at most skim them briefly.

Part Three outlines the construction and operation of a WfMS simulator. Chapter 11 deals with the process-oriented input data as well as simulation. Chapter 12 is devoted to the simulation of resources and the organizational settings required for the simulation study. Chapter 13 is concerned with the simulation experiments and the analysis of the simulation results. The last chapter emphasizes the general simulation phases that should be taken care in the simulation study to avoid misuse of a simulator.

The reader is expected to have some familiarity with programming concepts and background in probability and statistics as a prerequisite.

Hong Zhang
1 INTRODUCTION OF WORKFLOW MANAGEMENT SYSTEM

1.1 Basic Workflow Terminology

All organizations in the world have tasks or activities to do for obtaining some organization objectives. "In any organization, there are certain tasks that require information from several individuals. Information is collected, compiled and communicated as work moves through the organization until the task is completed. Workflow management is simply the automation of that movement of information to make the process more efficient." [Currid, 1994, p. 114]

"Workflow is concerned with the automation of procedures where documents, information or tasks are passed between participants according to a defined set of rules to achieve or contribute to, an overall business goal. Whilst workflow may be manually organized, in practice most workflow is normally organized within the context of an IT system to provide computerized support for the procedural automation." [Hollingsworth, 1995, p. 6] Here IT refers to the information technology.

The basic concepts with the relationships illustrated in Figure 1-1 are given

Figure 1-1. Relationships between Basic Workflow Concepts
by the Workflow Management Coalition.

- **Business process**: a “set of one or more linked procedures or activities which collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles and relationships.” [WfMC, 1996, p. 9]

- **Workflow**: the “automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.” [WfMC, 1996, p. 7]

- **Workflow management system** (WFMS): a “system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke appropriate IT tools and applications.” [WfMC, 1996, p. 8]

- **Process definition**: the “representation of a business process in a form which supports automated manipulation, such as modelling, or enactment by a workflow management system. The process definition consists of a network of activities and their relationships, criteria to indicate the start and termination of the process, and information about the individual activities, such as participants, associated IT applications and data, etc.” [WfMC, 1996, p. 10]

- **Activity**: a “description of a piece of work that forms one logical step within a process.” “A workflow activity requires human and/or machine resources(s) to support process execution; where human resource is required an activity is allocated to a workflow participant.” [WfMC, 1996, p. 11]

- **Organizational role**: a group of participants exhibiting a specific set of attributes, qualifications and/or skills. Typically any of the participants within a particular organizational role group can undertake an activity or work item requiring a resource with that set of attributes. See [WfMC, 1996, p. 47]

- **Instance**: the representation of a single enactment of a process (i.e. **process instance**), or activity (i.e. **activity instance**) within a process, including its associated data. Each instance represents a separate thread of execution of the process or activity, which may be controlled independently and will have its own internal state and externally visible identity, which may be used as a handle, for example, to record or retrieve audit data relating to the individual enactment. See [WfMC, 1996, p. 13]

  “A process instance is created, managed and (eventually) terminated by a workflow management system, in accordance with the process definition.” “Each process instance represents one individual enactment of the process, using its own process instance data, and which is
INTRODUCTION OF WORKFLOW MANAGEMENT SYSTEM

(normally) capable of independent control and audit as it progresses towards completion or termination. It represents the unit of work with respect to a business process that passes through a workflow management system (for example, the processing of one insurance claim, or the production of one engineering design).” [WfMC, 1996, p. 13]

“An activity instance is created and managed by a workflow management system when required within the enactment of process, in accordance with the process definition.” “Each activity instance represents a single invocation of an activity, relates to exactly one process instance and uses the process instance data associated with the process instance. Several activity instances may be associated with one process instance, where parallel activities exist within the process, but one activity instance cannot be associated with more than one process instance.” “Each activity instance is normally capable of independent control and audit and exhibits internal state.” [WfMC, 1996, p. 15]

• **Workflow participant**: a “resource which performs the work represented by a workflow activity instance. This work is normally manifested as one or more work items assigned to the workflow participant via the worklist.” “The term workflow participant is normally applied to a human resource but it could conceptually include machine-based resources such as an intelligent agent.” “A workflow participant may be identified directly within the business process definition, or (more normally) is identified by reference within the process definition to a role, which can then be filled by one or more of the resources available to the workflow management system to operate in that role during process enactment.” [WfMC, 1996, p. 16]

• **Work item**: the “representation of the work to be processed (by a workflow participant) in the context of an activity within a process instance.” [WfMC, 1996, p. 17]

• **Worklist**: a “list of work items associated with a given workflow participant (or in some cases with a group of workflow participants who may share a common worklist). The worklist forms part of the interface between a workflow engine and the worklist handler.” [WfMC, 1996, p. 18]

• **Invoked application**: “a workflow application that is invoked by the workflow management system to automate an activity, fully or in part, or to support a workflow participant in processing a work item.” [WfMC, 1996, p. 38]

• **Escalation**: a “procedure (automated or manual) which is invoked if a particular constraint or condition is not met.” [WfMC, 1996, p. 48]

**Build-time** and **run-time** refer to the time period before and after the implementation of a process definition respectively. Run-time is the time “when processes are executing or are to be executed.” [Lawrence, 1997, p. xxi]
The concepts belonging to process build-time determine those belonging to run-time.

1.2 The Espresso Workflow Management System

The Espresso WfMS is a document-oriented workflow application system that supports well-structured workflow processes as well as flexible and loosely structured processes—exception handling of routing and ad-hoc workflow are allowed in the system. The WfMS enables automation of key business processes by tracking and routing information and documents around in an organization.

Lotus Notes, a networked application that users located throughout the world can share the information organized in Notes databases, is the groupware platform for the Espresso WfMS. “Lotus Notes is an enterprise or workgroup computing environment that helps people work together effectively, regardless of platform or technical, organizational, geographical, or time-based boundaries. Lotus Notes based information can be shared across any distance, at any time.” [Toulemonde, 1998, p.19]

In the Espresso Application Database, Notes build-in workflow mechanisms are utilized to control flow of work items between workflow participants. Notes standard interfaces make it possible to integrate other IT applications of various media in the database. Notes fields with type of text, number, or time can be contained in a condition formula for routing a work. The workflow and application data are stored in Notes documents. Notes forms are used to display subsets of the document data, depending on the current state of the process instance. A Notes view can be designed for presenting worklists for different workflow participants. Scheduled or mail-triggered Notes agents can be assigned to workflow activities and act as workflow participants. For dynamic enactment and data verification, procedures written in LotusScript (a script language for Lotus Notes) can be included in the process definition. The Espresso workflow engine is not mail-based. The routing of a work from one activity to the next is accomplished in the databases itself. Through the Notes replication feature data can be synchronized between the distributed databases of the same Replica ID, the identification number of a database, and so it can be guaranteed that every workflow participant deals with the most up-to-date data. See [Kremer, 1999] for the technology of replication.

Furthermore, Notes internal access- and security mechanisms guarantees security at all stages of workflow management. Scaleable access control is provided through name directories, hierarchical access rights as well as through encryption and electronic signatures.

Database is “a file of interrelated data that are stored together to serve one or more applications and that are independent of programs using the data.” [Weinberg, 1980, p. 311] The Espresso WfMS mainly consists of three Lotus Notes databases:
• a real world *Application Database* where process instances are created and routed in accordance with process definitions;
• an *Organization Database* and/or a *Notes Organization Directory* (that is, Notes Address Book or Domino Directory) keeping an organizational model of human and material resources; and
• a *Process Database* holding the process definitions.

For the purpose of simulation, the Simulation Database is required for saving and retrieving simulation settings, simulation input data associated with process definitions and with resource databases, and simulation reports.

### 1.2.1 The ProcessModeler

The *ProcessModeler* (PM) is a graphical tool for process modeling, simulating and analyzing.

PM is an easy to use modeling tool, allowing process definitions to be modeled without the need for any programming. Simple point and click enables graphical creation and modification of activities and connections between activities. Specification of activities and connections are performed by completing the associated dialog boxes for specifying Notes forms to display documents representing activity instances, assigning an organizational role to an activity, defining routing conditions, and so on. PM can detect modeling errors, such as non-reachable activities, infinite loops, deadlocks, not available resources, etc.

Process definitions are stored in an Espresso Process Database. In the database, every process definition consists of a header document, a text document and a collection of activity documents—one activity document for each activity within the process definition respectively. Definitions of connections are saved in the documents of outgoing activities. Once completed in PM, a process definition can be immediately implemented and made available for use by the workflow participants to create process instances in an Application Database.

PM allows process definitions to be simulated or animated in the way they are enacted by the Espresso workflow engine. Process behaviors can be forecasted and potential bottlenecks can be detected before they are implemented. For a process definition that has been implemented in an Application Database, PM can read all running process instances in accordance with the process definition for analyzing and controlling flows of the work items in the WfMS.
PART ONE: PROCESS MODELING AND VERIFICATION

Figure 1-2 displays the relationships between PM functions and Lotus Notes databases in the Espresso WfMS.

PM is a process-oriented modeling tool. But with the help of activity groups, the process designer can define the network of a process definition in top-down or bottom-up approach. “Top-down design is a general strategy that specifies creating a system’s design in terms of its functions. Major functions are defined and then broken down into intermediate functions, which are broken down into detailed, lesser functions, and so on, until functions are sufficiently trivial to be implemented by a manageably small amount of code. Top-down design has the advantage of forcing the designer to consider the major functions (the most important modules) first and the less important ones later. It also forces the designer to consider the amount and nature of the code necessary to implement the design.” [Weinberg, 1980, p. 174] “In a large company with a relatively diversified group of businesses, ‘capacity limitations’ at the corporate level dictate a more or less bottom-up approach. The divisions initiate much of the goal setting, since it requires intimate knowledge of the industry-specific set of business conditions.” [Lorange, 1993, p. 25]

In PM and the Espresso WfMS, a process definition is called a process, a process instance a job, and an activity a task. An activity instance or a work associated with an activity instance is called a document.
1.2.2 Application Database

One or multiple Application Databases act as the run-time environment of the Espresso WfMS. The Application Database is the basic workflow enactment database where Notes documents representing activity and process instances are created or copied and routed from workflow participant to participant in accordance with the process definitions saved in a Process Database or through ad hoc workflow. A workflow participant must have authority to access the database. In addition to human resources, a scheduled or mail-triggered Notes agent defined in the Application Database can also be a workflow participant. Worklists of different workflow participants are categorized in a Notes view in the database.

A process instance can be created within the Application Database by a workflow participant belonging to the organizational role assigned to a start activity of a process definition connected to the Application Database. Creating a process instance in the Espresso WfMS means creating a document representing the instance of the process as well as the start activity. Workflow and application data are stored in the document.

Notes documents representing activity instances are allocated to the worklist of all workflow participants belonging to the organizational role assigned to the activity. A human workflow participant can execute activities presenting in his worklist. He can open a document and execute the activity. The description of the activity is displayed in a specified Notes form and the workflow participant may fill out the form as required. Sometimes it is necessary to invoke other applications automatically to give the current workflow participant the required information and tools for the execution of the activities. If a Notes agent is specified as a workflow participant, it will execute the activity at its’ scheduled time or when a mail comes in.

When an activity within a process instance is completed, the work associated with the process instance flows to the next activity according to the connections and routing options. The document representing the process instance now does not represent the instance of the completed activity but that of the next activity. The document is not sent to the next workflow participant via e-mail, but is allocated to the worklist of the workflow participants of the next activity. But the Notes internal e-mail system can be used to notify or to remind the workflow participants that they have new or urgent work to do.

If a split activity is completed and the work will flow to multiple next activities, the document representing the instance of split activity as well as the process instance will be copied to represent respectively each of the next activities. In this case, there are parallel work items associated with the same process instance in the Application Database.

Before the work at a join activity is allocated to worklists of relevant workflow participants for executing, the documents that are within the same process instance and represents different instances of the previous activities
1.2.3 Organizational Model

An Organization Database and/or a Notes Organization Directory comprise an organizational model for the Espresso WfMS. An Organizational model is “a model which represents organisational entities and their relationships; it may also incorporate a variety of attributes associated with the entities. Such a model may be realised in a directory or other form of database. Such a model normally incorporates concepts such as hierarchy, authority, responsibilities and attributes associated with an organisational role.” [WiMC, 1996, p. 47]

One of the following five organizational entities can act as an organizational role in the Espresso WfMS.

- **Person**: basic organization entity defined in an Organization Database and/or a Notes Organization Directory. A person is a human resource and can act as a workflow participant.
- **Notes group**: a team of human resources and/or IT resources defined in a Notes Organization Directory. A Notes group can include other groups as sub-groups. Only the groups containing human resources can be specified as organizational roles and only people in the group can act as workflow participants.
- **Workgroup**: a team of people defined in an Organization Database for a certain organizational objective or project. A member can be specified as the manager of a workgroup. A person can belong to different workgroups.
- **Department**: a team of people defined in an Organization Database. A department can include several other departments (called sub-departments of the department) but have only one parent department. A person can belong to only one department in an Organization Database. That is, departments in an Organization Database are hierarchically structured. A manager can be defined for a department.
- **Role**: (with or without role parameter) a team of people with certain attributes, qualifications, and/or skills. Roles are defined in an Organization Database A person can belong to different roles. Role parameters can be determined at run-time for specifying different workflow participants to the same activity under different situations.

For determination of workflow participants to execute an activity, an Organization Database and/or a Notes Organization Directory must be configured to the Application Database where process instances will be created.
1 INTRODUCTION OF WORKFLOW MANAGEMENT SYSTEM

at run-time. At process build-time, the databases can be used by PM to assign
defined organizational roles to activities.

1.3 Process Examples

PM presents the activity network of a process definition graphically in terms of
a directed network, or a directed graph (see [Eiselt, 1977], [Gribik, 1985] and
[Steward, 1981]). Activities are represented by icons and connected by directed
arcs, called links in PM. Links identify the flows of work items between
activities and are used to affect the desired activity execution thread—parallel
or sequential.

A process definition, the process instances in accordance with the process
definition, activities and links will be introduced in Example 1-1; a cycle of
activities in Example 1-2; and parallel, split, and join activities in Example 1-3.

Example 1-1

In Figure 1-3, process definition “Order” consists of four activities:
“Register order”, “Check order”, “Complete order” and “Notification”.
The activities (represented by icons) are connected by four links
(represented by directed arcs).

![Figure 1-3. Process Definition “Order”](image)

At run-time, a process instance is created at activity “Register order”,
which is the start activity of the process definition and marked with a
flag. Activity “Check order” can be executed after completion of activity
“Register order”. Activity “Complete order” will be executed only when
an order is accepted after execution of activity “Check order”. After
completion of activity “Notification”, the process instance will be
terminated.

Activity execution thread of a process instance generated at run-time
in accordance with the process definition can be either:
1. Register order → Check order → Notification; or
2. Register order → Check order → Complete order → Notification.

Example 1-2
Process definition “Report” presented in Figure 1-4 contains five activities. Opposite links between activity “Work on report” and activity “Proofread report” build up a cycle. Both activities will be executed repetitively until no corrections are required.

Process instances generated according to the process definition can be seen as activity execution threads as:

1. Request report → Work on report → Proofread report → Send document → Receive report;
2. Request report → Work on report → Proofread report → Work on report → Proofread report → Send document → Receive report; or

Figure 1-4. Process Definition “Report”

“Proofread report” build up a cycle. Both activities will be executed repetitively until no corrections are required.

Process instances generated according to the process definition can be seen as activity execution threads as:

Example 1-3
Process definition “Loan” as shown in Figure 1-5 consists of seven activities. In this process definition, if applied loan is a high value, activities “Check personal creditworthiness” and “Check asset valuations” can be executed simultaneously. That is, a process instance may include multiple concurrent execution threads. Activities “Check
1 INTRODUCTION OF WORKFLOW MANAGEMENT SYSTEM

... personal creditworthiness and "Check Asset valuations" are called parallel activities.

Because after execution of activity "Evaluate", a single execution thread may split into two parallel execution threads, activity "Evaluate" is a split activity. Before execution of activity "Approve credit", multiple concurrent execution threads of a process instance converge into a single execution thread. So activity "Approve credit" is a join activity. Before execution of a join activity, synchronization may be required for joining parallel work items from previous activities.

1.4 Conclusion

The most important concepts used all over this work are Workflow Management System (WfMS), process definition, activity, process instance, activity instance, organization role, workflow participant, worklist and work item.

The WfMS enable the automation of business processes. A process instance is an automated business process that is created and routed in accordance with the process definition, a network of activities. The activities within a process definition is assigned to an organization role and the activity within a process instance is executed by workflow participants which are resolved from the organization role. For each workflow participant, there is a worklist for allocating the work items associated with the process instances.

The ProcessModeler (PM), belonging to the Espresso WfMS, is a product where the process definition, verification and the simulation model discussed in...
this work are implemented, except those sections explicitly marked with asterisk ("*”).

In the following chapters, a work item will be simply called a *work*. 
2 PROCESS DEFINITION

A process definition, denoted by \((T, L, \text{Sources}(T))\), consists of a finite and non-empty set of activities, denoted by \(T\), \(T \neq \emptyset\), a collection of links connecting certain pairs of activities, denoted by \(L\), and a non-empty set of start activities, denoted by \(\text{Sources}(T)\), \(\text{Sources}(T) \subseteq T\).

2.1 Activity and Link

Activity \(i, i \in T\), is an indivisible piece of work within a process definition that will be performed by human and/or IT resources at run-time. Here \(i\) is an identical number of activities within a process definition, i.e. \(\partial i, j \in T\), with \(i \neq j\).

Activities are represented graphically as nodes or vertex of a network and are displayed with icons by PM (see Figure 1-3, Figure 1-4 and Figure 1-5).

A link with direction from activity \(j\) to activity \(k\), \(j \in T, k \in T - \{j\}\), displayed by PM with a directed arc as shown in Figure 2-1, is denoted by \((j, k)\). Link \((j, k), (j, k) \in L\), connects activity \(j\) to activity \(k\). Activity \(j\) is the origin of the link and activity \(k\) is the destination of the link. Activity \(k\) is called a successor of activity \(j\), and activity \(j\) is called a predecessor of activity \(k\).

Links determine execution order of activities in a process definition. At runtime a link makes the destination activity possible to be invoked for execution when the origin activity is completed (see Chapter 3).

Example 2-1
The process definition presented in Figure 2-2 consists of four activities,
i.e.

\[ T = \{1, 2, 3, 4\}, \]

and six links, i.e.

\[ L = \{(1, 3), (2, 1), (2, 3), (2, 4), (3, 2), (3, 4)\}. \]

**Assumption 2-1. Parallel Links**

There are no parallel links in a process definition, i.e. \( \partial(i, j) \in L \) and \( \partial(k, q) \in L \),
- if \( i = k \), then \( j \neq q \);
- if \( j = q \), then \( i \neq k \).

Parallel links are not allowed to be created in PM. That is, if there is a link connecting activity \( i \) to activity \( j \), it is not possible to create another link from activity \( i \) to activity \( j \).

The set of outgoing links of activity \( j \), denoted by \( T^O(j) \), is defined as

\[ \{(j, k) \mid \forall k \in T \text{ with } (j, k) \in L\}. \]

The set of incoming links, denoted by \( pT(j) \), is defined as

\[ \{(k, j) \mid \forall k \in T \text{ with } (k, j) \in L\}. \]

**Example 2-2**

For the process definition in Figure 2-2,

\[ T^O(1) = \{(1, 3)\}, \]
\[ T^O(2) = \{(2, 1), (2, 3), (2, 4)\}, \]
\[ T^O(3) = \{(3, 2), (3, 4)\}, \]
\[ T^O(4) = \emptyset, \]
\[ pT(1) = \{(2, 1)\}, \]
\[ pT(2) = \{(3, 2)\}, \]
\[ pT(3) = \{(1, 3), (2, 3)\}, \]
\[ pT(4) = \{(2, 4), (3, 4)\}. \]

**2.2 Start Activity**

*Start activity* \( s \in \text{Sources}(T) \), is a specified activity where a process instance can be created at run-time. A start activity is represented in PM with a flag
above the activity icon. For the process definition in Figure 2-2, only activity 2 is specified as a start activity, i.e.

\[ \text{Sources}(T) = \{2\}. \]

If \( p_T(i) = \emptyset \) (that is, activity \( i \) has no predecessor), activity \( i \) is a *structural start activity*.

The following rules are used for specifying start activities of a process definition:

- A structural start activity should be specified as a start activity of a process definition. For example, in Figure 2-3, both activity 3 and activity 10 have no predecessor. They are therefore structural start activities and must be start activities of the process definition. Set \( p_T(i), i \in T \), is used to determine whether activity \( i \) is a structural start activity.
- One activity involved in a structural start cycle (see Chapter 4), such as activity 7 or activity 8 in Figure 2-3, should be specified as a start activity of the process definition.
- Any activity in a process definition can be specified as a start activity. For example in Figure 2-3, activity 1 is specified voluntarily as a start activity of the process definition.

**Assumption 2-2.** Creation of a Process Instance

A process instance can be created at run-time only at one of the start activities of a process definition.

According to Assumption 2-2, if a process definition has no start activity, no process instance associated with the process definition can be created. So such a process definition is not allowed.

**Assumption 2-3.** Start Activities

A process definition must have at least one start activity, i.e. \( \text{Sources}(T) \neq \emptyset \).
PM verifies Assumption 2-3 (see the algorithm for Determining Potential Start Activities in Chapter 4), when preparing to implement a process definition (saving as an executable version).

### 2.3 Algorithms for Keeping a Process Definition

The following algorithms are utilized for keeping data of the activity network of a process definition \((T, L, \text{Sources}(T))\). Data of the activity network are used in almost all algorithms in the process modeling and verification.

**Hypothesis**

\(T, L\) and \(\text{Sources}(T)\) represent respectively sets of activities, links, and start activities of a process definition. \(T^\delta(i)\) and \(P^\delta(i), \forall i \in T\), represent sets of outgoing links and incoming links of activity \(i\) respectively. These sets represent a state of structural definitions during process modeling and vary with the changes of the network of a process definition.

**Principle**

Data of the activity network of a process definition are updated with the changes to the structure of the process definition.

**Procedures**

1. When creating a new process definition:

   \[
   T \leftarrow \emptyset; \\
   L \leftarrow \emptyset; \\
   \text{Sources}(T) \leftarrow \emptyset.
   \]

2. When creating new activity \(i\) in a process definition:

   \[
   T \leftarrow T \cup \{i\}; \\
   T^\delta(i) \leftarrow \emptyset; \\
   P^\delta(i) \leftarrow \emptyset.
   \]

3. When creating new link \((j, k)\) in a process definition:

   \[
   L \leftarrow L \cup \{(j, k)\}; \\
   T^\delta(j) \leftarrow T^\delta(j) \cup \{(j, k)\}; \\
   P^\delta(k) \leftarrow P^\delta(k) \cup \{(j, k)\}.
   \]
4. When removing link \((j, k)\) from a process definition:

\[
L \leftarrow L - \{(j, k)\}; \\
T^S(j) \leftarrow T^S(j) - \{(j, k)\}; \\
P^T(k) \leftarrow P^T(k) - \{(j, k)\}.
\]

5. When removing activity \(i\) from a process definition:

\[
T \leftarrow T - \{i\}; \\
Sources(T) \leftarrow Sources(T) - \{i\}; \\
\text{Call removing link \((i, n)\) from a process definition, } \forall (i, n) \in T^S(i); \\
\text{Call removing link \((p, i)\) from a process definition, } \forall (p, i) \in P^T(i).
\]

1. When specifying activity \(i\) as a start activity of a process definition:

\[
Sources(T) \leftarrow Sources(T) \cup \{i\}.
\]

7. When specifying that activity \(i\) is no more a start activity of a process definition:

\[
Sources(T) \leftarrow Sources(T) - \{i\}.
\]

### 2.4 Conclusion

The network of a process definition (that is, the sets of activities, links and start activities) are determined by the process designer. Unlike the start activities, where a process instance can be created indicated by the designer, end activities, where a process instance can be terminated, are not specified by the designer, but are determined according to the set of outgoing links of the activity as well as routing options of the links (see Chapter 3).

The structural state of a modeling process definition includes:

- the set of activities, denoted by \(T\);
- the set of links, denoted by \(L\);
- the set of start activities, denoted by Sources\((T)\);
- the set of outgoing links of an activity, denoted by \(T^S(i), i \in T\); and
- the set of incoming links of an activity, denoted by \(P^T(i), i \in T\).

Figure 2-4 illustrates the relationships between them.
Figure 2-4. Keeping a Process Definition
At run-time, sequential as well as parallel routing occurs within a process instance. The following are the definitions given by the Workflow Management Coalition.

- **Sequential routing**: a “segment of a process instance under enactment by a workflow management system, in which several activities are executed in sequence under a single thread of execution. (No -split or -join conditions occur during sequential routing.).” [WfMC, 1996, p. 27]

- **Parallel routing**: “a segment of a process instance under enactment by a workflow management system, where two or more activity instances are executing in parallel within the workflow, giving rise to multiple threads of control.” [WfMC, 1996, p. 26]

The set of links and routing options of each link in a process definition together determine sequential or parallel routing of a process instance in the Espresso WfMS.

In addition, routing options of the outgoing links of an activity determine whether the activity is an end activity of a process definition.

### 3.1 Routing Option Definition

The *routing option* of a link in the context of a process definition determines at run-time whether to route a work along the link. When the origin activity of a link is completed, the work associated with the instance of the process definition is sent out from the origin activity of the link and the activity instance is eliminated. If the work can be routed to the destination of the link according to the routing option, an activity instance of the destination will be created. When the workflow participants of the destination activity are invoked by the work coming from a predecessor activity (that is, the work is allocated in their worklists), they can perform the destination activity of the link.

One of five routing options “Always”, “Multiple Choice”, “Exclusive Choice”, “Condition” and “Else” can be defined to a link. Suppose that link \((j, k)\) is specified with one of the following routing options, when a work associated with the process instance at activity \(j\) is completed,

- **Always**: the work will always be routed to the destination of the link (activity \(k\));
- **Multiple Choice**: the workflow participant who completes the origin activity of the link (activity \(j\)) can choose one or more of all outgoing “Multiple Choice” links of the origin activity (i.e. choose from set \(\{(j, k)\}\))
\( n \) with that \((j, n)\) is a “Multiple Choice” link). Along each selected link, the work will be routed to the destination of the link. If multiple links are selected, multiple activity instances will be created after eliminating the instance of activity \( j \).

- **Exclusive Choice**: the workflow participant who completes the origin activity of the link (activity \( j \)) should choose one and only one of all outgoing “Exclusive Choice” links of the origin activity (i.e. choose one from set \( \{ (j, n) \mid \forall (j, n) \in T^S(j) \text{ with that } (j, n) \text{ is an “Exclusive Choice” link} \} \)). The work will be routed along the selected link to its destination.

- **Condition**: the work will be routed to the destination of the link (activity \( k \)), if a condition is met. The condition is described by a logic formula that can be evaluated by the workflow engine.

- **Else**: the work will be routed to the destination of the link (activity \( k \)), if the work can not be routed along one of any other outgoing links of the origin activity (i.e. if not \( \exists (j, n) \in T^S(j) - \{(j, k)\} \) for which the work can be routed along \((j, n)\) to activity \( n \)).

An “Else” link ensures that the work associated with a process instance flows further from the origin activity of the link and a process instance do not terminate at the activity. It makes sense when it is an outgoing link of an activity that has merely “Multiple Choice” or “Condition” links as other outgoing links of the activity. That is, if \( \exists (i, k) \in T^S(i) \text{ for which } (i, k) \text{ is an “Else” link, } \forall (i, n) \in T^S(i) - \{(i, k)\}, (i, n) \text{ is either a “Multiple Choice” or a “Condition” link.}

“Multiple Choice” and “Exclusive Choice” links must be decided by people at run-time for whether to route a work associated with a process instance to the destinations of the links or not. PM does not allow workflow participants of the origin activities of these kinds of links being IT resources (see the algorithm for Getting Invalid IT Resource Activities in Section 3.2).

Routing option of a link is defined through the dialog box as shown in Figure 3-1. For “Multiple Choice” and “Exclusive Choice” links, descriptions of the links should be given in order to help people at run-time to choose routing links after completing the origin activity. For a “Condition” link, a condition formula evaluated to logic value TRUE or FALSE must be given. The result of the formula determines whether a work associated with a process instance flows along the link at run-time.

According to the routing option definitions, an activity is a routing decision activity if one of its outgoing links is not an “Always” link. The decision about whether to route a work further from a routing decision activity is made either by the workflow participant who completes it, or automatically by the workflow engine of the WfMS.

In the examples of process maps, the routing option of a link can be recognized by either the routing option indicated beside or on the link represented by a solid arc, or by the drawing style of the link as compared in Figure 3-2. A solid arc stands for an “Always” link, a dashed arc an “Exclusive
Choice” link, a dash-dotted a “Multiple Choice” link or a “Condition” link, and a dotted arc an “Else” link. A solid arc without routing option indication represents an “Always” link. In other words, if a link is drawn with a solid arc but it is not an “Always” link, the routing option of the link is indicated.
3.2 Algorithm for Getting Invalid IT Resource Activities

This algorithm is used for verifying a process definition.

**Hypothesis**

$T$ represents the activity set of a process definition. $T^S(i)$, $\forall i \in T$, denotes the set of outgoing links of activity $i$.

**Principle**

“Multiple Choice” and “Exclusive Choice” links must be decided by people at run-time. Therefore it has to be verified that origin activities of these links are not assigned to IT resources (such as Notes agents). A set of invalid defined activities will be returned by the procedure.

Temporary set InvalidActivities keeps invalid activities, set RestActivities keeps activities that have not be dealt with, and RestLinks keeps not considered outgoing links of the current activity.

**Procedure**

Step 1: InvalidActivities $\leftarrow \phi$;
Step 2: RestActivities $\leftarrow T$;
Step 3: if RestActivities = $\phi$, go to Step 12;
Step 4: remove an element, say activity $i$, from RestActivities;
Step 5: if no workflow participant of activity $i$ is an IT resource, go to Step 3;
Step 6: RestLinks $\leftarrow T^S(i)$;
Step 7: if RestLinks = $\phi$, go to Step 3;
Step 8: remove an element, say link $(i, n)$, from RestLinks;
Step 9: if link $(i, n)$ is “Multiple Choice” or “Exclusive Choice”, go to Step 11;
Step 10: go to Step 7;
Step 11: InvalidActivities $\leftarrow$ InvalidActivities $\cup \{i\}$; go to Step 3;
Step 12: stop (return InvalidActivities).

3.3 End Activity

An *end activity* of a process definition is an activity within a process definition, where a work associated with a instance of the process definition can be terminated or stopped after completion of the activity at run-time. End activities are determined by the structural definition of a process definition. The set of end activities of a process definition is denoted by $\text{Sinks}(T)$, $\text{Sinks}(T) \subseteq T$.

A *structural end activity* is an activity where a work associated with a process instance in accordance with the process definition is possible to be terminated in a WfMS. Activity $i$ is a structural end activity, if
1. $T^S(i) = \phi$; or
2. $\forall (i, n) \in T^S(i)$, with that $(i, n)$ is a “Multiple Choice” or “Condition” link.

For a structural end activity $i$ with $T^S(i) \neq \phi$, activity $i$ is a routing decision activity and can be specified as a non-end activity, i.e. let $i \notin \text{Sinks}(T)$. Thus at run-time a work associated with a process instance can not be stopped at activity $i$—it will flow further soon after one of “Multiple Choice” links in set $T^S(i)$ is selected or a formula in one of “Condition” links is valued to TRUE.

**Example 3-1**

In Figure 3-3, activities 4, 5, 6 and 7 are structural end activities which must belong to set $\text{Sinks}(T)$, because they have no successor, i.e. $T^S(4) = \phi$, $T^S(5) = \phi$, $T^S(6) = \phi$, and $T^S(7) = \phi$.

Activity 3 is a structural end activity that can be specified as a non-end activity of the process definition, since all its outgoing links (i.e. link (3, 6) and link (3, 7)) are “Condition” links. If activity 3 is specified as a non-end activity, i.e. $3 \notin \text{Sinks}(T)$, after execution of activity 3, a work associated with a process instance will wait there till $X > 100$ or $X > 1000$. If activity 3 is an end activity, a work associated with a process instance can stop there when it is being completed with $X \leq 100$.

Activity 2 is a structural end activity and can be specified as a non-end activity too, for all its outgoing links (i.e. link (2, 4) and link (2, 5)) are “Multiple Choice” links. If Activity 2 is specified as a non-end activity, the person who completes activity 2 should choose at least one link from links “East” and “West”; otherwise, a work associated with a process instance will stop there if the person who completes the activity does not choose any link for further routing.
3.4 Conclusion

A link connecting two activities can be one of five routing options of “Always”, “Multiple Choice”, “Exclusive Choice”, “Condition” and “Else”. Routing options make it possible to route a work along a link under certain decisions or conditions at run-time.

Whether to route a work along a “Multiple Choice” or “Exclusive Choice” link is decided by the workflow participants who complete the origin activity. So it will be verified that such kinds of routing options are not included in the outgoing links of an activity performed by IT resources.

The end activities of a process definition, denoted by Sinks(T), are determined mainly by the network of activities as well as routing options of the outgoing links of an activity.

Figure 3-4 presents, from which process definition data, invalid IT resource activities and end activities are determined.
4 PATH AND CYCLE

4.1 Paths

In a process definition, *paths* from activity \( i \) to activity \( k \), denoted by \( i \rightarrow k \), exists if one of the following recursive definitions holds:

1. \((i, k) \in L\); or
2. \(\exists q \in T\), for which \( \exists i \rightarrow q \) and \((q, k) \in L\).

If \( \exists i \rightarrow k \), it can be said that activity \( i \) has a path to activity \( k \), or activity \( k \) is reachable from activity \( i \), or there are paths from activity \( i \) to activity \( k \).

A certain path of \( i \rightarrow k \) can be denoted by an alternating sequence of activities and links as

\[(i, (i, q_1), q_1, (q_1, q_2), q_2, \ldots, q_n, (q_n, k), k),\]

or simply by a sequence of activities as

\[(i, q_1, q_2, \ldots, q_n, k).\]

Here \( q_1, q_2, \ldots, q_n, i, k \in T \) and \((i, q_1), (q_1, q_2), \ldots, (q_n, k) \in L\).

According to the path definition, a path must consist of at least one link, say \((i, k)\), and the path can be denoted by \((i, (i, k), k)\) or simply by \((i, k)\), the same denotation as that of the link.

A path of \( i \rightarrow k \) implies that activity \( i \) precedes (or affects) activity \( k \) and a work at activity \( i \) has the potential to flow to activity \( k \). An instance of activity \( i \) may curse creating instance of activity \( k \).

**Theorem 4-1. Path Transitivity**

If \( i \rightarrow k \) and \( k \rightarrow j \) exist, then \( i \rightarrow j \) exists.

**Proof**

1) because \( \exists i \rightarrow k \), so

\[\exists q_1, q_2, \ldots, q_n, i, k \in T \text{ and } (i, q_1), (q_1, q_2), \ldots, (q_n, k) \in L \text{ with } (i, (i, q_1), q_1, (q_1, q_2), q_2, \ldots, q_n, (q_n, k), k);\]

2) because \( \exists k \rightarrow j \), so

\[\exists p_1, p_2, \ldots, p_m, k, j \in T \text{ and } (k, p_1), (p_1, p_2), \ldots, (p_m, j) \in L \text{ with } (k, (k, p_1), p_1, (p_1, p_2), p_2, \ldots, p_m, (p_m, j), j);\]

3) from results of 1) and 2), we get

\[(i, (i, q_1), q_1, (q_1, q_2), q_2, \ldots, q_n, (q_n, k), k, (k, p_1), p_1, (p_1, p_2), p_2, \ldots, p_m, (p_m, j), j);\]
That is, a path of $i \to j$ exists.

End of proof

Theorem 4-1 supports the algorithms discussed later for determining paths.

**Example 4-1**

![Figure 4-1. Path](image)

In the process definition shown in Figure 4-1, there are five links, i.e.

$$L = \{(1, 2), (2, 3), (2, 5), (3, 5), (5, 2)\}.$$  

According to the first path definition, we know that there exist paths

$$1 \to 2, \ 2 \to 3, \ 2 \to 5, \ 3 \to 5 \text{ and } 5 \to 2.$$  

Now according to the second path definition, from paths

- $1 \to 2, (2, 3)$;
- $1 \to 2, (2, 5)$;
- $2 \to 3, (3, 5)$;
- $2 \to 5, (5, 2)$;
- $3 \to 5, (5, 2)$;
- $5 \to 2, (2, 3)$; and
- $5 \to 2, (2, 5)$

respectively, the following paths exist:

- $1 \to 3$ (i.e. $(1, 2, 3)$),
- $1 \to 5$ (i.e. $(1, 2, 5)$),
- $2 \to 5$ (i.e. $(2, 3, 5)$),
- $2 \to 2$ (i.e. $(2, 5, 2)$),
- $3 \to 2$ (i.e. $(3, 5, 2)$),
- $5 \to 3$ (i.e. $(5, 2, 3)$), and
- $5 \to 5$ (i.e. $(5, 2, 5)$).

Again according to the second definition, from paths

- $(1, 2, 3), (3, 5)$;
- $(1, 2, 5), (5, 2)$;
respectively, we know that there exist paths:

1→5 (i.e. (1, 2, 3, 5)),
1→2 (i.e. (1, 2, 5, 2)),
2→2 (i.e. (2, 3, 5, 2)),
2→3 (i.e. (2, 5, 2, 3)),
2→5 (i.e. (2, 5, 2, 5)),
3→3 (i.e. (3, 5, 2, 3)),
3→5 (i.e. (3, 5, 2, 5)),
5→5 (i.e. (5, 2, 3, 5)), and
5→2 (i.e. (5, 2, 5, 2))

Furthermore, we can determine other paths of the process definition.

### 4.1.1 Elementary Path

A path of $i \rightarrow k$ is called **elementary**, if all activities on the path appear only once, except that the beginning activity can also be the ending activity of a path. That is, $(i, q_1, q_2, ..., q_n, k)$ is an elementary path if

$$\forall j \in [1, n] \text{ with } i \neq q_j, k \neq q_j, \text{ and } \forall j, f \in [1, n] \text{ and } j \neq f, \text{ with } q_j \neq q_f$$

The set of activities on an elementary path of $i \rightarrow k$ is denoted by $\text{Activities}(i \rightarrow k)$.

For example, in Example 4-1 we have got two different paths for 2→3:

(2, 3), and
(2, 5, 2, 3)

Elementary path of 2→3 is (2, 3), and

$$\text{Activities}(2 \rightarrow 3) = \{2, 3\}.$$
An elementary path of \( i \rightarrow k \) gives a way of a work flowing from activity \( i \) to activity \( k \) over activities within set Activities(\( i \rightarrow k \)).

**Theorem 4-2.** Elementary Path

If a path of \( i \rightarrow k \) exists, an elementary path of \( i \rightarrow k \) exists too.

**Proof**

1) Because \( \exists i \rightarrow k \), so

\[ \exists q_1, q_2, ..., q_n, i, k \in T \text{ and } (i, q_1), (q_1, q_2), ..., (q_n, k) \in L \text{ with } \]

\[ (i, i, q_1), (q_1, q_2), q_2, ..., q_n, (q_n, k), k); \]

2) if \( i \neq q_j \) and \( k \neq q_j \) \((j = 1, 2, ..., n)\) and \( q_1, q_2, ..., q_n \) are different from one another (i.e. \( \forall j \neq f, j, f \in [1, n] \text{ with } q_j \neq q_f \)), \( i \rightarrow k \) is an elementary path;

3) suppose only \( i = q_j, j = 1, 2, ..., n \), then we have

\[ (q_i, (q_j + 1, q_j + 2), q_j + 2, ..., q_n, (q_n, k), k) \text{ or } \]

\[ (i, (q_j + 1, q_j + 2), q_j + 2, ..., q_n, (q_n, k), k). \]

That is, an elementary path of \( i \rightarrow k \) exists;

4) suppose only \( k = q_j, j = 1, 2, ..., n \), then exists

\[ (i, (i, q_1), q_1, ..., q_{j-1}, (q_{j-1}, q_j), q_j) \text{ or } \]

\[ (i, (i, q_1), q_1, ..., q_{j-1}, (q_{j-1}, q_j), k). \]

That is, an elementary path of \( i \rightarrow k \) exists;

5) suppose only \( q_j = q_j, j = 1, 2, ..., n \), and \( j < f \) path

\[ (i, (i, q_1), (q_1, q_2), q_2, ..., q_n, (q_n, k), k) \]

can then be denoted as

\[ (i, (i, q_1), q_1, ..., (q_{j-1}, q_j), q_j, (q_j, q_{j+1}), ..., (q_{f-1}, q_f), q_f, (q_f, q_{f+1}), ..., q_n, (q_n, k), k); \]

because \( q_j = q_f \), so exists path

\[ (i, (i, q_1), q_1, ..., (q_{j-1}, q_j), q_j, (q_j, q_{j+1}), ..., q_n, (q_n, k), k), \]

that is, an elementary path of \( i \rightarrow k \).

From 3), 4) and 5) we know that any non-elementary path of \( i \rightarrow k \) can be transformed to an elementary path of \( i \rightarrow k \).

End of Proof

From this theorem, it is known that if no elementary path of \( i \rightarrow k \) exists, no path of \( i \rightarrow k \) exists either.

**4.1.2 Algorithm for Finding a Path**

This algorithm is, given the set of outgoing links of each activity, to find a path between two activities within a process definition without passing over other specific activities.
This algorithm is not used in process verification and simulation, because performance problems arise to find all paths between any two different activities in a process definition. It can be implemented in PM, if some relevant features that do not require all paths are developed in the modeling tool.

**Hypothesis**

$T$ denotes the set of activities within a process definition. $T^S(i), i \in T,$ stands for the set of outgoing links of activity $i$.

**Principle**

Activity $i$, activity $k$ and set $U$ are parameters of the main procedure. Here $i \neq k$, $U \neq \emptyset$, and $U \subseteq T$. This procedure will return a certain path of $i \rightarrow k$, if there exists a path from activity $i$ to activity $k$ that does not include any activities in set $U$; otherwise return empty.

To determine whether such a path exists, a self-called sub procedure with parameter $q$ is used here. Activity $q$ is the activity from which the path till activity $k$ will be further searched. The path built up during searching is kept in stack $\text{PathStack}(j), j = 1, 2, \ldots, \text{StackPointer}$. When the sub procedure is called by the main procedure, $\text{StackPointer}$ is assigned with 0, thus activity $i$, the beginning activity of the path, is kept in $\text{PathStack}(1)$.

Temporary set $\text{RestLinks}$ is used in the sub procedure for keeping the not treated links. Variable $\text{CurPath}$ keeps the path.

**Main Procedure** ($i, k, U$)

Step 1: if $i \in U$ or $k \in U$, stop (return FALSE);
Step 2: $\text{StackPointer} \leftarrow 0$;
Step 3: $\text{CurPath} \leftarrow \text{call the sub procedure with parameter } i$;
Step 4: stop (return $\text{CurPath}$).

**Sub Procedure** ($q$)

Step 1: $\text{StackPointer} \leftarrow \text{StackPointer} + 1$, $\text{PathStack}(\text{StackPointer}) \leftarrow q$ (put $q$ in stack);
Step 2: if $q = k$ (paths $i \rightarrow k$ exist), stop (return activity series of $\text{PathStack}(1)$, $\text{PathStack}(2), \ldots, \text{PathStack}(\text{StackPointer})$);
Step 3: $\text{RestLinks} \leftarrow T^S(q)$;
Step 4: if $\text{RestLinks} = \emptyset$, go to Step 10;
Step 5: remove an element, say link $(q, n)$, from set $\text{RestLinks}$;
Step 6: if $n \in U$ (activity $n$ belongs to set $U$ and cannot be on the path), go to Step 4;
Step 7: if $n = \text{PathStack}(j)$ with $j \in [1, \text{StackPointer}]$ (activity $n$ is on current searching path), go to Step 4;
Step 8: $\text{CurPath} \leftarrow \text{call the sub procedure self with parameter } n$; If $\text{CurPath}$ is not empty (paths $i \rightarrow q \rightarrow n \rightarrow k$ exist), stop (return $\text{CurPath}$);
Step 9: go to Step 4;
Step 10: (paths $q \rightarrow k$ don’t exist) StackPointer $\leftarrow$ StackPointer $-$ 1 (remove $q$ from PathStack());
Step 11: stop (return empty).

4.2 Reachability and Criticality

4.2.1 Reachability

A set of activities which are reachable from activity $i$ is denoted by $Reaches(i)$. That is, $k \in Reaches(i)$, if a path of $i \rightarrow k$ exists.

Example 4-2
From Example 4-1, it is known that in the process definition shown in Figure 4-1 there exist paths:

$$\begin{align*}
&1 \rightarrow 2, 2 \rightarrow 3, 2 \rightarrow 5, 3 \rightarrow 5, 5 \rightarrow 2; \\
&1 \rightarrow 3, 1 \rightarrow 5, 2 \rightarrow 5, 2 \rightarrow 2, 3 \rightarrow 2, 5 \rightarrow 3, 5 \rightarrow 5; \\
&1 \rightarrow 5, 1 \rightarrow 2, 2 \rightarrow 2, 2 \rightarrow 3, 2 \rightarrow 5, 3 \rightarrow 3, 3 \rightarrow 5, 5 \rightarrow 5, 5 \rightarrow 2; \text{ etc.}
\end{align*}$$

Thus:

Reaches(1) = $\{2, 3, 5\}$, Reaches(2) = $\{2, 3, 5\}$,
Reaches(3) = $\{2, 3, 5\}$, Reaches(5) = $\{2, 3, 5\}$.

Theorem 4-3. Reachability
If $k \in Reaches(i)$ and $i \in Reaches(p)$, then $k \in Reaches(p)$.

Proof
Because $k \in Reaches(i)$ and $i \in Reaches(p)$, so exist paths of $i \rightarrow k$ and $p \rightarrow i$.
According to Theorem 4-1, exists a path of $p \rightarrow k$.
That is, $k \in Reaches(p)$.
End of Proof

If an activity is not reachable from any start activity of a process definition, the activity will have no chance to be executed at run-time. So it is nonsense to define such an activity in a process definition. PM is able to verify the following assumption.
Assumption 4-1. Activity Reachability
All activities in a process definition must be reachable from a start activity.
That is, $\forall i \in T, \exists s \in \text{Sources}(T)$, with $i \in \text{Reaches}(s)$.

### 4.2.2 Criticality

In a process definition, multiple elementary paths of $i \rightarrow k$ may exist. That is, a work associated with an instance of the process definition may take different ways flowing from activity $i$ to activity $k$. The set of all elementary paths $i \rightarrow k$ in a process definition is denoted by $\text{Paths}(i \rightarrow k)$.

According to the Theorem 4-2, if a path of $i \rightarrow k$ exists, $\text{Paths}(i \rightarrow k) \neq \emptyset$. For example, in Figure 4-1 of Example 4-1, there are two different elementary paths for $2 \rightarrow 5$:

- $(2, 5)$, and
- $(2, 3, 5)$.

So $\text{Paths}(2 \rightarrow 5) = \{(2, 5), (2, 3, 5)\}$.

Activity $p, p \in T - \{i, k\}$, is called a critical activity on paths $i \rightarrow k$, if

$$\forall i \rightarrow k \in \text{Paths}(i \rightarrow k) \text{ with } p \in \text{Activities}(i \rightarrow k).$$

The definition implies that $p \neq i$ and $p \neq k$. That is, beginning activity $i$ and ending activity $k$ of the paths are not included in critical activities of paths $i \rightarrow k$.

If activity $p$ is a critical activity on paths $i \rightarrow k$, it must be executed if a work flows from activity $i$ to activity $k$. The set of critical activities of paths $i \rightarrow k$ is denoted by $\text{Criticals}(i \rightarrow k)$.

**Example 4-3**

For the process definition shown in Figure 4-1,

<table>
<thead>
<tr>
<th>Path</th>
<th>Criticals</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 \rightarrow 1$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$1 \rightarrow 2$</td>
<td>${(1, 2)}$</td>
</tr>
<tr>
<td>$1 \rightarrow 3$</td>
<td>${(1, 2, 3)}$</td>
</tr>
<tr>
<td>$1 \rightarrow 5$</td>
<td>${(1, 2, 3, 5), (1, 2, 5)}$</td>
</tr>
<tr>
<td>$2 \rightarrow 1$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$2 \rightarrow 2$</td>
<td>${(2, 5, 2), (2, 3, 5, 2)}$</td>
</tr>
<tr>
<td>$2 \rightarrow 3$</td>
<td>${(2, 3)}$</td>
</tr>
<tr>
<td>$2 \rightarrow 5$</td>
<td>${(2, 5), (2, 3, 5)}$</td>
</tr>
<tr>
<td>$3 \rightarrow 1$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$3 \rightarrow 2$</td>
<td>${(3, 5, 2)}$</td>
</tr>
<tr>
<td>$3 \rightarrow 1$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$3 \rightarrow 2$</td>
<td>${(3, 5, 2)}$</td>
</tr>
</tbody>
</table>
That is, activity 2 is a critical activity of paths 1\to 3, 1\to 5, 3\to 3, 5\to 3 and 5\to 5; Activity 5 is a critical activity of paths 2\to 2, 3\to 2 and 3\to 3.

4.2.3 Algorithm for Determining Reachability and Criticality

The procedure is called by the algorithm for Determining Workflow Control Data in Chapter 5.

Hypothesis

\( T \) and \( L \) represent respectively sets of activities and links of a process definition. \( \text{Reaches}(i), \forall i \in T \), denotes a set of activities reachable from activity \( i \). \( \text{Criticals}(i \to k), \forall i, k \in T \), stands for the set of critical activities of paths \( i \to k \).

Principle

According to the link set of a process definition, \( \text{Reaches}(i), i \in T \), are determined. \( \text{Criticals}(i \to k) \) will also be updated, \( \forall k \in \text{Reaches}(i) \).

Temporary set \( \text{Previous}(i), \forall i \in T \), keeps all activities that have paths to activity \( i \). Temporary set \( \text{RestLinks} \) keeps links that have not been treated.

Procedure

Step 1: \( \text{Reaches}(i) \leftarrow \emptyset \) and \( \text{Previous}(i) \leftarrow \emptyset, \forall i \in T \);

Step 2: \( \text{Criticals}(i \to k) \leftarrow T, \forall i, k \in T \);

Step 3: \( \text{RestLinks} \leftarrow L \);

Step 4: if \( \text{RestLinks} = \emptyset \), go to Step 13;

Step 5: remove an element, say link \((i, k)\), from \( \text{RestLinks} \);

Step 6: \( \forall p \in \text{Previous}(i) \), let

\[
\text{Reaches}(p) \leftarrow \text{Reaches}(p) \cup \{k\} \cup \text{Reaches}(k);
\]

\[
\text{Criticals}(p \to k) \leftarrow \text{Criticals}(p \to k) \cap (\text{Criticals}(p \to i) \cup \{i\});
\]

\[
\text{Criticals}(p \to s) \leftarrow \text{Criticals}(p \to s) \cap (\text{Criticals}(p \to i) \cup \{i\} \cup \{k\} \cup \text{Criticals}(k \to s)), \forall s \in \text{Reaches}(k);
\]

Step 7: \( \forall n \in \text{Reaches}(k) \), let

\[
\text{Previous}(n) \leftarrow \text{Previous}(n) \cup \{i\} \cup \text{Previous}(i);
\]

Step 8: \( \text{Reaches}(i) \leftarrow \text{Reaches}(i) \cup \{k\} \cup \text{Reaches}(k) \);
Step 9: \( \text{Previous}(k) \leftarrow \text{Previous}(k) \cup \{i\} \cup \text{Previous}(i) \);

Step 10: \( \text{Criticals}(i \rightarrow k) \leftarrow \emptyset \) \( (\text{link } (i, k) \text{ makes no critical activity existing on paths of } i \rightarrow k) \);

Step 11: \( \forall s \in \text{Reaches}(k), \) let

\[ \text{Criticals}(i \rightarrow s) \leftarrow \text{Criticals}(i \rightarrow s) \cap (\{k\} \cup \text{Criticals}(k \rightarrow s)) \]

Step 12: go to Step 4;

Step 13: \( \text{Criticals}(i \rightarrow k) \leftarrow \emptyset \), \( \forall i, k \in T \) with \( k \notin \text{Reaches}(i) \);

Step 14: stop \( (\text{Reaches}(i), \text{Criticals}(i \rightarrow k), \forall i \in T \text{ and } \forall k \in \text{Reaches}(i), \text{have been determined}) \).

Example 4-4
If the algorithm is used for determining reachable and critical activities for the process definition shown in Figure 4-1, at the first time at Step 4, the values are initialized as:

<table>
<thead>
<tr>
<th>( i )</th>
<th>( 1 )</th>
<th>( 2 )</th>
<th>( 3 )</th>
<th>( 5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Reaches}(i) )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( \text{Previous}(i) )</td>
<td>( \emptyset )</td>
<td>( {1} )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \text{Criticals}(i \rightarrow k) )</th>
<th>( i = 1 )</th>
<th>( i = 2 )</th>
<th>( i = 3 )</th>
<th>( i = 5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( k = 1 )</td>
<td>{1, 2, 3, 5}</td>
<td>{1, 2, 3, 5}</td>
<td>{1, 2, 3, 5}</td>
<td>{1, 2, 3, 5}</td>
</tr>
<tr>
<td>( k = 2 )</td>
<td>{1, 2, 3, 5}</td>
<td>{1, 2, 3, 5}</td>
<td>{1, 2, 3, 5}</td>
<td>{1, 2, 3, 5}</td>
</tr>
<tr>
<td>( k = 3 )</td>
<td>{1, 2, 3, 5}</td>
<td>{1, 2, 3, 5}</td>
<td>{1, 2, 3, 5}</td>
<td>{1, 2, 3, 5}</td>
</tr>
<tr>
<td>( k = 5 )</td>
<td>{1, 2, 3, 5}</td>
<td>{1, 2, 3, 5}</td>
<td>{1, 2, 3, 5}</td>
<td>{1, 2, 3, 5}</td>
</tr>
</tbody>
</table>

\( \text{RestLinks} = \{(1, 2), (2, 3), (2, 5), (3, 5), (5, 2)\} \).

After dealing with link \((1, 2)\), and going back to Step 4, the values become:

<table>
<thead>
<tr>
<th>( i )</th>
<th>( 1 )</th>
<th>( 2 )</th>
<th>( 3 )</th>
<th>( 5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Reaches}(i) )</td>
<td>{2}</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( \text{Previous}(i) )</td>
<td>( \emptyset )</td>
<td>{1}</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \text{Criticals}(i \rightarrow k) )</th>
<th>( i = 1 )</th>
<th>( i = 2 )</th>
<th>( i = 3 )</th>
<th>( i = 5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( k = 1 )</td>
<td>{1, 2, 3, 5}</td>
<td>{1, 2, 3, 5}</td>
<td>{1, 2, 3, 5}</td>
<td>{1, 2, 3, 5}</td>
</tr>
<tr>
<td>( k = 2 )</td>
<td>( \emptyset )</td>
<td>{1, 2, 3, 5}</td>
<td>{1, 2, 3, 5}</td>
<td>{1, 2, 3, 5}</td>
</tr>
<tr>
<td>( k = 3 )</td>
<td>{1, 2, 3, 5}</td>
<td>{1, 2, 3, 5}</td>
<td>{1, 2, 3, 5}</td>
<td>{1, 2, 3, 5}</td>
</tr>
<tr>
<td>( k = 5 )</td>
<td>{1, 2, 3, 5}</td>
<td>{1, 2, 3, 5}</td>
<td>{1, 2, 3, 5}</td>
<td>{1, 2, 3, 5}</td>
</tr>
</tbody>
</table>

\( \text{RestLinks} = \{(2, 3), (2, 5), (3, 5), (5, 2)\} \).

After treating link \((2, 3)\), and going back to Step 4, the values become:
RestLinks = \{ (2, 5), (3, 5), (5, 2) \}.

After processing link (2, 5), and going back to Step 4, the values become:

\[
\begin{array}{c|cccc}
  \text{i} & 1 & 2 & 3 & 5 \\
  \text{Reaches}(i) & \{2, 3\} & \{3\} & \varnothing & \varnothing \\
  \text{Previous}(i) & \varnothing & \{1\} & \{2, 1\} & \varnothing \\
\end{array}
\]

\[
\begin{array}{c|cccc}
  \text{i} & 1 & 2 & 3 & 5 \\
  \text{Criticals}(i \rightarrow k) & \text{i = 1} & \text{i = 2} & \text{i = 3} & \text{i = 5} \\
  k = 1 & \{1, 2, 3, 5\} & \{1, 2, 3, 5\} & \{1, 2, 3, 5\} & \{1, 2, 3, 5\} \\
  k = 2 & \varnothing & \{1, 2, 3, 5\} & \{1, 2, 3, 5\} & \{1, 2, 3, 5\} \\
  k = 3 & \{2\} & \varnothing & \{1, 2, 3, 5\} & \{1, 2, 3, 5\} \\
  k = 5 & \{1, 2, 3, 5\} & \{1, 2, 3, 5\} & \{1, 2, 3, 5\} & \{1, 2, 3, 5\} \\
\end{array}
\]

RestLinks = \{ (3, 5), (5, 2) \}.

After considering link (3, 5), and going back to Step 4, the values become:

\[
\begin{array}{c|cccc}
  \text{i} & 1 & 2 & 3 & 5 \\
  \text{Reaches}(i) & \{2, 3, 5\} & \{3, 5\} & \varnothing & \varnothing \\
  \text{Previous}(i) & \varnothing & \{1\} & \{2, 1\} & \varnothing \\
\end{array}
\]

\[
\begin{array}{c|cccc}
  \text{i} & 1 & 2 & 3 & 5 \\
  \text{Criticals}(i \rightarrow k) & \text{i = 1} & \text{i = 2} & \text{i = 3} & \text{i = 5} \\
  k = 1 & \{1, 2, 3, 5\} & \{1, 2, 3, 5\} & \{1, 2, 3, 5\} & \{1, 2, 3, 5\} \\
  k = 2 & \varnothing & \{1, 2, 3, 5\} & \{1, 2, 3, 5\} & \{1, 2, 3, 5\} \\
  k = 3 & \{2\} & \varnothing & \{1, 2, 3, 5\} & \{1, 2, 3, 5\} \\
  k = 5 & \{2\} & \varnothing & \{1, 2, 3, 5\} & \{1, 2, 3, 5\} \\
\end{array}
\]

RestLinks = \{ (3, 5), (5, 2) \}.

After handling link (5, 2), and going back to Step 4, the values become:

\[
\begin{array}{c|cccc}
  \text{i} & 1 & 2 & 3 & 5 \\
  \text{Reaches}(i) & \{2, 3, 5\} & \{3, 5, 2\} & \{5, 2, 3\} & \{2, 3, 5\} \\
  \text{Previous}(i) & \varnothing & \{1, 5, 2, 3\} & \{2, 1, 5, 3\} & \{2, 1, 3, 5\} \\
\end{array}
\]
Criticals\((i \rightarrow k)\) | \(i = 1\) | \(i = 2\) | \(i = 3\) | \(i = 5\) \\
--- | --- | --- | --- | --- \\
k = 1 | \{1, 2, 3, 5\} | \{1, 2, 3, 5\} | \{1, 2, 3, 5\} | \{1, 2, 3, 5\} \\
k = 2 | \phi | \{5\} | \{5\} | \phi \\
k = 3 | \{2\} | \phi | \{5, 2\} | \{2\} \\
k = 5 | \{2\} | \phi | \phi | \phi \\

RestLinks = \(\phi\);

Now because \(\text{RestLinks} = \phi\), go to Step 13, where Criticals\((1 \rightarrow 1)\), Criticals\((2 \rightarrow 1)\), Criticals\((3 \rightarrow 1)\) and Criticals\((5 \rightarrow 1)\) are set to \(\phi\), for \(1 \notin \text{Reaches}(1)\), \(1 \notin \text{Reaches}(2)\), \(1 \notin \text{Reaches}(3)\), and \(1 \notin \text{Reaches}(4)\). Eventually reachable and critical values are:

\[\begin{array}{c|cccc}
   i & 1 & 2 & 3 & 5 \\
--- & --- & --- & --- & --- \\
Reaches(i) & \{2, 3, 5\} & \{3, 5, 2\} & \{5, 2, 3\} & \{2, 3, 5\} \\
\end{array}\]

\[\begin{array}{c|cccc}
   \text{Criticals}(i \rightarrow k) & i = 1 & i = 2 & i = 3 & i = 5 \\
--- & --- & --- & --- & --- \\
k = 1 & \phi & \phi & \phi & \phi \\
k = 2 & \phi & \{5\} & \{5\} & \phi \\
k = 3 & \{2\} & \phi & \{5, 2\} & \{2\} \\
k = 5 & \{2\} & \phi & \phi & \phi \\
\end{array}\]

They are the same as the results in Example 4-2 and Example 4-3.

### 4.3 Cycles

A cycle, or circuit called by [Steward, 1981], is a path of \(i \rightarrow k\), for which \((k, \hat{i}) \in L\). That is, a cycle is a path with the beginning activity being a successor of the ending activity of the path. A cycle is called elementary, if all activities on the cycle appear only once.

**Example 4-5**

In Figure 4-1 of Example 4-1, since \((5, 2), (2, 3), (2, 5), (3, 5) \in L\), the following paths obtained in Example 4-3 are element cycles of the process definition:

\[(2, 5), (2, 3, 5), (3, 5, 2), (5, 2)\] and \[(5, 2, 3)\].
Note that there are only two elementary cycles in the process definition. Cycles (2, 5) and (5, 2) are the same cycle denoted with different beginning activity. That is,

\[
\text{cycle (2, 5)} \equiv \text{cycle (5, 2)}.
\]

Similarly,

\[
\text{cycle (2, 3, 5)} \equiv \text{cycle (3, 5, 2)} \equiv \text{cycle (5, 2, 3)}.
\]

A cycle is an *infinite cycle*, if a work on the cycle can neither depart from the cycle nor be stopped on the cycle.

### Assumption 4.2. Infinite Cycle
No infinite cycle is allowed in a process definition to ensure that any process instance created at run-time can eventually terminate or stop at one end activity of the process definition.

If there is an infinite cycle in a process definition, a process instance created in accordance with the process definition may never be terminated or stopped. PM will confirm that there are no infinite cycles in a process definition (see the algorithm for Getting Infinite Elementary Cycles in Section 4.3.3).

### 4.3.1 Algorithm for Determining Activity Reachable to End

This algorithm is called by the algorithm for Getting Infinite Elementary Cycles (see Section 4.3.3).

**Hypothesis**

*\( T \)* denotes the activity set of a process definition. *Sinks\((T)\)* represents the set of end activities of the process definition. *ToEnd\((i)\) (= TRUE or FALSE), \( \forall i \in T, \)* stands for whether activity \( i \) has a path to an end activity of the process definition. *\( T^a(i) \)* is the set of outgoing links of activity \( i \).

**Principle**

This procedure will determine *ToEnd\((i)\), \( \forall i \in T \).*

To determine *ToEnd\((i)\),* a self-called sub procedure is used. Here activity \( i \) is a parameter of the sub procedure. The path built up during searching is kept in stack *PathStack\((j)\), \( j = 1, 2, ..., \)*, *StackPointer*. When the sub procedure is called by the main procedure, *StackPointer* is assigned with 0. The activities on the path have no path to an end activity.

*ToEnd\((i)\)* will be assigned with TRUE, if \( \exists s \in \text{Sinks}(T) \) with activity \( i \) has a path to activity \( s \). In the meantime, *ToEnd\((n)\), \( \forall (i, n) \in T^a(i) \),* will also be
determined. The number of newly determined activities NewlyDetermined will be returned by the sub procedure. It is assigned to 0, when the sub procedure is called by the main procedure.

When dealing with activity $i$ in the sub procedure, data of outgoing links of activity $i$ is kept in the following temporary variables:

- ALs, XLs and ELs keep respectively numbers of “Always”, “Exclusive Choice” and “Else” links; if activity $i$ is specified as a non-end activity, ELs will be set as a non-zero value;
- ALEnds and XLEnds are respectively used for keeping numbers of “Always” and “Exclusive Choice” links, whose destination activities have paths to an end activity of the process definition;
- TotalLEnds keeps the total number of all different kinds of links whose destination activities have a path to one of the end activities of the process definition.
- ALNils and XLNils are respectively used for keeping numbers of “Always” and “Exclusive Choice” links, whether whose destination activities have paths to an end activity of the process definition can not be determined; and
- TotalLNils keeps the total number of all different kinds of links, whether whose destination activities have a path to one of the end activities of the process definition can not be determined.

According to the values of above variables, at Steps 13, 14 and 15 of the sub procedure with parameter $i$, ToEnd$(i)$ can be determined.

Temporary variable TotalDetermined and sets RestActivities and RestLinks are also used in the procedure.

Main Procedure
Step 1: clear ToEnd$(i)$ to empty; $\forall i \in T$;
Step 2: TotalDetermined $\leftarrow 0$; RestActivities $\leftarrow T$;
Step 3: if RestActivities $=$ $\emptyset$, go to Step 8;
Step 4: remove an element, say activity $i$, from RestActivities;
Step 5: if ToEnd$(i)$ is not empty (ToEnd$(i)$ as well as ToEnd$(n)$, $\forall (i, n) \in T^S(i)$, has been determined), go to Step 3;
Step 6: StackPointer $\leftarrow 0$; NewlyDetermined $\leftarrow 0$; call the sub procedure with parameter $i$ (NewlyDetermined may be updated);
Step 7: TotalDetermined $\leftarrow$ TotalDetermined + NewlyDetermined; go to Step 3;
Step 8: if TotalDetermined $\neq 0$ (some activities have been determined reachable to an end activity), go to Step 2 (try again for all not determined);
Step 9: stop (no more possible to determine ToEnd$(i)$, $\forall i \in T$).
Sub Procedure \((i)\)

Step 1: if \(\mathcal{T}(i) = \emptyset\) (activity \(i\) has no outgoing link. So it is a structural end activity and can not be specified as a non-end activity), let \(\text{ToEnd}(i) \leftarrow \text{TRUE}\) and stop;

Step 2: \(\text{StackPointer} \leftarrow \text{StackPointer} + 1; \text{PathStack}(\text{StackPointer}) \leftarrow i;\)

Step 3: \(\text{ALs} \leftarrow 0; \text{XLs} \leftarrow 0; \text{ELs} \leftarrow 0;\)

Step 4: \(\text{ALEnds} \leftarrow 0; \text{XLEnds} \leftarrow 0; \text{TotalLEnds} \leftarrow 0;\)

Step 5: \(\text{ALNils} \leftarrow 0; \text{XLNils} \leftarrow 0; \text{TotalLNils} \leftarrow 0;\)

Step 6: \(\text{RestLinks} \leftarrow \mathcal{T}(i);\)

Step 7: remove an element, say link \((i, \, n)\), from set \(\text{RestLinks};\)

Step 8: if \((i, \, n)\) is an “Always” link, then \(\text{ALs} \leftarrow \text{ALs} + 1;\)

if \((i, \, n)\) is an “Exclusive Choice” link, then \(\text{XLs} \leftarrow \text{XLs} + 1;\)

if \((i, \, n)\) is an “Else” link, then \(\text{ELs} \leftarrow \text{ELs} + 1;\)

Step 9: if \(\text{ToEnd}(n)\) is not empty (activity \(n\) as well as its successors have been dealt with), go to Step 13;

Step 10: if \(n = \text{PathStack}(j)\) with \(j \in [1, \text{StackPointer}]\) (activity \(n\) is on the searching path), go to Step 12;

Step 11: call the sub procedure \(\text{self}\) with parameter \(n;\)

Step 12: if \(\text{ToEnd}(n)\) is empty (whether activity \(n\) has a path to an end activity can not be determined),

1° \(\text{TotalLNils} \leftarrow \text{TotalLNils} + 1;\)

2° if \((i, \, n)\) is an “Always” link, then \(\text{ALNils} \leftarrow \text{ALNils} + 1;\)

if \((i, \, n)\) is an “Exclusive Choice” link, then \(\text{XLNils} \leftarrow \text{XLNils} + 1;\)

3° go to Step 14;

Step 13: if \(\text{ToEnd}(n) = \text{TRUE},\)

1° \(\text{TotalLEnds} \leftarrow \text{TotalLEnds} + 1;\)

2° if \((i, \, n)\) is an “Always” link, then \(\text{ALEnds} \leftarrow \text{ALEnds} + 1;\)

if \((i, \, n)\) is an “Exclusive Choice” link, then \(\text{XLEnds} \leftarrow \text{XLEnds} + 1;\)

3° go to Step 14;

Step 14: if \(\text{RestLinks} \neq \emptyset\), go to Step 7;

Step 15: if \(\text{ALs} = 0, \text{XLs} = 0, \text{ELs} = 0,\) and \(i \notin \text{Sinks}(T)\) (activity \(i\) is specified as a non-end activity), let \(\text{ELs} \leftarrow \text{ELs} + 1\)

Step 16: if \(\text{ALs} = \text{ALEnds},\) go to Step 18;

Step 17: (at least one “Always” link has no path to an end activity)

if \(\text{ALNils} > 0\) (ToEnd\((i)\) can not be determined), go to Step 24;

else (a work at activity \(i\) can never be terminated)

let \(\text{ToEnd}(i) \leftarrow \text{FALSE}\) and go to Step 23;

Step 18: if \(\text{XLs} = 0\) or \(\text{XLEnds} > 0\), go to Step 20;

Step 19: (at least along one of the “Exclusive Choice” links the work associated with a process instance flows further, but none of the destinations of the links have paths to an end activity)

if \(\text{XLNils} > 0\), go to Step 24;
else let ToEnd($i$) $\leftarrow$ FALSE and go to Step 23;

Step 20: if ELs = 0 or TotalLEnds > 0, go to Step 22;
Step 21: (“Else” link confirms that the work associated with a process definition should flow further, but no successor of activity $i$ has paths to an end activity)
  if TotalLNils > 0, go to Step 24;
  else let ToEnd($i$) $\leftarrow$ FALSE and go to Step 23;

Step 22: ToEnd($i$) $\leftarrow$ TRUE;

Step 23: NewlyDetermined $\leftarrow$ NewlyDetermined + 1;

Step 24: StackPointer $\leftarrow$ StackPointer − 1; stop.

4.3.2 Algorithm for Getting Elementary Cycles

This algorithm is called by the algorithm for Getting Infinite Elementary Cycles (see Section 4.3.3) and by the algorithm for Determining Potential Start Activities (see Section 4.4).

Hypothesis
Sets $T$ and $L$ represent respectively the activity set and the link set of a process definition. Set $T^S(i)$, $\forall i \in T$, denotes the set of outgoing links of activity $i$.

Principle
The set of elementary cycles of a process definition kept in set Cycles will be returned.

If link $(k, i)$ exists, the self-called sub procedure with parameters $i$, $k$ and $U$ is called for getting all elementary paths of $i \rightarrow k$ connected by links in set $U$. Current path built up during searching is kept in stack PathStack($j$), $j = 1, 2, \ldots$, StackPointer. When the sub procedure is called by the main procedure, StackPointer is assigned with 0. The found path and link $(k, i)$ combine a cycle and will be added to set Cycles.

Temporary set RestLinks is used in the procedure.

Main Procedure
Step 1: call Determining Reachability and Criticality (Reaches($i$), $i \in T$, will be determined);

Step 2: Cycles $\leftarrow \phi$;

Step 3: RestLinks $\leftarrow L$;

Step 4: if $|\text{RestLinks}| \leq 1$ (links in set RestLinks can not combine any cycle), stop (return Cycles);

Step 5: remove an element, say link $(k, i)$, from set RestLinks;

Step 6: StackPointer $\leftarrow 0$; call the sub procedure with parameters $i$, $k$ and RestLinks;

Step 7: go to Step 4.
Sub Procedure \((i, k, U)\)

Step 1: StackPointer \(\leftarrow\) StackPointer + 1, PathStack(StackPointer) \(\leftarrow i;\)

Step 2: if \(i = k\) (the path exists), add sequence of activities on the searching path (PathStack(1), PathStack(2), ..., PathStack(StackPointer)) to set Cycles, and go to Step 10;

Step 3: RestLinks \(\leftarrow T^S(i);\)

Step 4: if RestLinks = \(\emptyset\), go to Step 10;

Step 5: remove an element, say link \((i, n)\), from set RestLinks;

Step 6: if \((i, n) \notin U\), go to Step 4;

Step 7: if \(k \notin \text{Reaches}(n)\) (activity \(n\) has no path to activity \(k\)), go to Step 4;

Step 8: if \(n = \text{PathStack}(j)\) with \(j \in [1, \text{StackPointer}]\) (activity \(n\) is on current searching path), go to Step 4;

Step 9: call the sub procedure self with parameters \(n, k\) and \(U\); go to Step 4;

Step 10: StackPointer \(\leftarrow\) StackPointer – 1; stop.

4.3.3 Algorithm for \textbf{Getting Infinite Elementary Cycles}\n
This algorithm is used for verification of a process definition.

Hypothesis

\(T\) denotes the set of activities within a process definition. Variable ToEnd\((i), i \in T,\) stands for whether activity \(i\) has a path to an end activity of the process definition.

Principle

All infinite elementary cycles of a process definition will be found and put in the returned set InfiniteCycles.

Temporary set RestCycles is used to keep not treated cycles; CurCycle keeps the sequence of activities that makes up a cycle.

Procedure

Step 1: call \textbf{Determining Activity Reachable to End} (ToEnd\((i), i \in T,\) will be updated);

Step 2: RestCycles \(\leftarrow\) \textbf{Getting Elementary Cycles};

Step 3: InfiniteCycles \(\leftarrow\) \(\emptyset\);

Step 4: if RestCycles = \(\emptyset\), stop (return InfiniteCycles);

Step 5: remove a cycle, say CurCycle, from RestCycles;

Step 6: if there is one activity on cycle CurCycle, say activity \(q,\) with that ToEnd\((q) = \text{TRUE},\) cycle CurCycle is not infinite; otherwise add cycle CurCycle to set InfiniteCycles;

Step 7: go to Step 4.
Example 4-6
For the process definition in Figure 4-2, all links are “Always” links. After performing the procedure of this algorithm, PM discovers four infinite elementary cycles as shown in Figure 4-3.

When an infinite cycle is selected in the dialog box, it will be indicated on the process map by square-marked links that make up the cycle. Figure 4-4 presents the four different infinite cycles of the process definition in Figure 4-2:

- Cycle (1, 2)
- Cycle (1, 2, 3)
- Cycle (1, 4, 2)
- Cycle (1, 4, 2, 3)

![Figure 4-2. Cycle](image)

![Figure 4-3. Verification Results—Infinite Cycle](image)

![Figure 4-4. Infinite Cycles](image)
4.4 Algorithm for Determining Potential Start Activities

This algorithm is used for verification of a process definition to ensure that all activities in a process definition are reachable from a start activity.

Hypothesis

$T$ denotes the activity set of a process definition. Sources($T$) stands for the set of start activities. $^pT(i)$, $i \in T$, represents the set of incoming links of activity $i$. Updated Reaches($i$), $\forall i \in T$, is a set of activities which are reachable from activity $i$.

Principle

A set of potential start activities will be returned by the procedure. It is a minimal set of non-start activities that should be specified as start activities so that all activities in a process definition are reachable from either a start activity or a potential start activity.

An activity without predecessors but not specified as a start is a potential start activity. An activity in an elementary cycle may also be a potential start activity.

Temporary set NotSpecified is used to keep the returned value. Set RestCycles keeps elementary cycles that have not been treated and set CycleStarts keeps the elementary cycles that may be structural start cycles.

Procedure

Step 1: NotSpecified $\leftarrow \emptyset$;
Step 2: $\forall i \in T$, if $^pT(i) = \emptyset$ and $i \notin$ Sources($T$) (activity $i$ has no predecessor but is not specified as a start activity), let NotSpecified $\leftarrow$ NotSpecified $\cup \{i\}$;
Step 3: CycleStarts $\leftarrow \emptyset$;
Step 4: RestCycles $\leftarrow$ Getting Elementary Cycles;
Step 5: if RestCycles $= \emptyset$, go to Step 11;
Step 6: remove a cycle, say $i \rightarrow k$, from RestCycles;
Step 7: if $\exists s \in$ Sources($T$) $\cup$ NotSpecified with $i \in$ Reaches($s$) (activity $i$ is reachable from a specified or potential start activity), go to Step 5;
Step 8: if $\exists s \rightarrow q \in$ CycleStarts with $i \in$ Reaches($s$) (activity $i$ is reachable from a cycle in set CycleStarts), go to Step 5;
Step 9: $\forall s \rightarrow q \in$ CycleStarts, if $s \in$ Reaches($i$) (cycle $s \rightarrow q$ is reachable from activity $i$), let CycleStarts $\leftarrow$ CycleStarts $\setminus \{s \rightarrow q\}$;
Step 10: CycleStarts $\leftarrow$ CycleStarts $\cup \{i \rightarrow k\}$; go to Step 5;
Step 11: NotSpecified $\leftarrow$ NotSpecified $\cup \{i\}$, $\forall i \rightarrow k \in$ CycleStarts;
Step 12: stop (return NotSpecified).
Example 4-7

For the process definition in Figure 4-5, activity 1 is specified as a start activity. Set \{3, 10, 8\} will be returned by the procedure. Here activity 3 and activity 10 have no predecessor but are not specified as start activities. The cycle consisting of activity 7 and activity 8 is a structural start cycle and one of the activities in the cycle, here activity 8, is determined as a potential start activity.

The dialog box “Verification Results” shown in Figure 4-6 informs that there are three disconnected process parts beginning with activities 3, 10 and 8 respectively and that all activities must be connected and be reachable from a start activity of a process definition.

![Verification Results](image)

**Figure 4-6. Verification Results—Start Activity**
4.5 Conclusion

Paths from activity $i$ to activity $k$, denoted by $i \rightarrow k$, are defined to explain whether activity $k$ is reachable from activity $i$. That is, if a path of $i \rightarrow k$ exists, a work at activity $i$ has the potential to flow to activity $k$.

All non-start activities within a process definition should be reachable from a start activity. The potential start activities are the minimal set of activities that should be further specified as the start activities of a process definition.

An activity $p$ is critical on paths of $i \rightarrow k$, denoted by $p \in \text{Criticals}(i \rightarrow k)$, if without go over activity $p$, activity $k$ is not reachable from activity $i$.

The set of activities which are reachable from activity $i$ is denoted by $\text{Reaches}(i)$. The sets $\text{Reaches}(i)$ and $\text{Criticals}(i \rightarrow k)$, $\forall i, k \in T$, are determined by an algorithmic procedure.

A cycle is a special path of $i \rightarrow k$, for which $(k, i) \in L$. On an element cycle all activities appear only once. Infinite cycles cause a work on them can never be terminated. So they should not exist in a process definition. Infinite cycles are detected through checking whether an activity on the cycle, say activity $i$, has a path to an end activity, denoted by $\text{ToEnd}(i)$.

Figure 4-7 presents from which process definition data, potential start activities, infinite elementary cycles are determined, and how the data and algorithms discussed in this chapter are related to one another.

![Diagram](image-url)
5 SPLIT AND JOIN

A split activity causes parallel routing of work items associated with a process definition. Parallel routing work items will be automatically joined at a join activity in the Espresso WfMS. The determination of join activities by PM decides where work items of parallel routing are joined at run-time.

5.1 Split Activity

Activity $i$ is a split activity of a process definition, if

1. $|T^\delta(i)| > 1$;
2. $\exists(i, n) \in T^\delta(i), (i, n)$ is not an “Exclusive Choice” link; and
3. when $|T^\delta(i)| = 2$, $\forall(i, n) \in T^\delta(i), (i, n)$ is not an “Else” link.

A split activity has multiple outgoing links. In other words, multiple links possess a split activity as the common origin. When the work associated with a process instance flows out from a split activity, it is possible that the work is routed to different successors of the split activity and multiple activity instances are created simultaneously. A split activity causes one execution thread of a process instance to split to multiple concurrent execution threads and parallel activity instances of a process instance exist in a WfMS.

The set of split activities of a process definition is denoted by $\text{Splits}(T)$, $\text{Splits}(T) \subset T$. If $\text{Splits}(T) \neq \emptyset$, it is possible that at run-time multiple activity instances are associated with a common process instance.

Example 5-1

In Figure 5-1, activity 1 is a split activity, because:

1. $T^\delta(1) = \{(1, 2), (1, 3)\}$ (i.e. $|T^\delta(1)| = 2$); and
2. neither link (1, 2) nor link (1, 3) is “Exclusive Choice” or “Else”.

Activity 1 has activity 2 and activity 3 as successors. After activity 1 is completed at run-time, the work associated with a process instance is routed to activity 2 and activity 3 simultaneously and then two activity instances within the process instance exist.
Example 5-2
In Figure 5-2, the two links named with “Europe” and “America” respectively are both “Exclusive Choice” links.

**Figure 5-2. Not a Split Activity—Exclusive Links**

Although activity 1 has two outgoing links, i.e. $T^s(1) = \{(1, 2), (1, 3)\}$, it is not a split activity, because $\partial(1, k)\in T^s(1)$, $(1, k)$ is an “Exclusive Choice” link. When activity 1 is completed at run-time, only one of the links $(1, 2)$ and $(1, 3)$ can be selected to let the work be routed along it.

Example 5-3

**Figure 5-3. Not a Split Activity—Else Link**

In Figure 5-3, link “Great quantity” is a “Condition” link and link “Small Quantity > 100” Activity 1 is not a split activity, because $|T^s(1)| = 2$, and $(1, 3)$ is an “Else” link. After activity 1 is completed, the work is routed from activity 1 to either activity 2 (if Quantity > 100) or activity 3, but not to both.

### 5.1.1 Algorithm for Updating Split Activities

This procedure is called by the algorithm for Determining Workflow Control Data (see Section 5.4.2).
Hypothesis

$T$ denotes the set of activities within a process definition. $T^*(i)$, $\forall i \in T$, represents the set of outgoing links of activity $i$. $\text{Splits}(T)$, $\text{Splits}(T) \subset T$, stands for the set of split activities of the process definition.

Principle

$\text{Splits}(T)$ will be determined according to the definition of a split activity. Temporary sets $\text{RestActivities}$, and $\text{RestLinks}$ are used here.

Procedure

Step 1: $\text{Splits}(T) \leftarrow \phi$

Step 2: $\text{RestActivities} \leftarrow T$

Step 3: remove an element, say activity $i$, from set $\text{RestActivities}$;

Step 4: if $|T^*(i)| < 2$ (activity $i$ has no or only one outgoing link), go to Step 12;

Step 5: if $|T^*(i)| > 2$, go to Step 7;

Step 6: (activity $i$ has two outgoing links) get the two links from $T^*(i)$, say $(i, n)$ and $(i, k)$; if $(i, n)$ or $(i, k)$ is an “Else” link, go to Step 12;

Step 7: $\text{RestLinks} \leftarrow T^*(i)$;

Step 8: remove an element, say $(i, n)$, from set $\text{RestLinks}$;

Step 9: if $(i, n)$ is not an “Exclusive Choice” link, go to Step 11;

Step 10: if $\text{RestLinks} = \phi$ (all outgoing links of activity $i$ are “Exclusive Choice” links), go to Step 12; otherwise go to Step 8;

Step 11: $\text{Splits}(T) \leftarrow \text{Splits}(T) \cup \{i\}$;

Step 12: if $\text{RestActivities} \neq \phi$, go back to Step 3; otherwise stop.

5.1.2 Ancestor of a Split Activity

For a given start activity $s$, $s \in \text{Sources}(T)$, split activity $p$ is called an ancestor of split activity $i$, $i \neq s$, if

$$p \in \{s\} \cup \text{Criticals}(s \rightarrow i)$$

The set of ancestors of split activity $i$ from start activity $s$ is denoted by $\text{PreSplits}(i \mid s)$. That is,

$$\text{PreSplits}(i \mid s) = (\{s\} \cup \text{Criticals}(s \rightarrow i)) \cap \text{Splits}(T), \text{ for } i \neq s;$$

$$\text{PreSplits}(i \mid s) = \phi, \text{ for } i = s.$$

Split activity $i$ is not reachable from start activity $s$ without going over all of the split activities in $\text{PreSplits}(i \mid s)$. 
Example 5-4

All links in the process definition shown in Figure 5-4 are “Always” links. We have:

\[
\begin{align*}
\text{Sources}(T) &= \{1\}; \\
\text{Splits}(T) &= \{1, 2, 3, 6\}; \\
\text{Criticals}(1\rightarrow 2) &= \emptyset, \\
\text{Criticals}(1\rightarrow 3) &= \emptyset, \\
\text{Criticals}(1\rightarrow 6) &= \{3\}; \\
\text{PreSplits}(1 \mid 1) &= \emptyset, \\
\text{PreSplits}(2 \mid 1) &= (\{1\} \cup \text{Criticals}(1\rightarrow 2)) \cap \text{Splits}(T) = \{1\}, \\
\text{PreSplits}(3 \mid 1) &= (\{1\} \cup \text{Criticals}(1\rightarrow 3)) \cap \text{Splits}(T) = \{1\}, \\
\text{PreSplits}(6 \mid 1) &= (\{1\} \cup \text{Criticals}(1\rightarrow 6)) \cap \text{Splits}(T) = \{1, 3\}.
\end{align*}
\]

That is, relevant to start activity 1, split activity 1 has no ancestor; split activity 1 is ancestor of split activity 2 and activity 3; both split activity 1 and activity 3 are ancestors of split activity 6.

5.1.3 Algorithm for Getting Ancestors of a Split Activity

This procedure is called by the algorithm for Getting Ancestor Joins (see Section 5.2.2).

Hypothesis
Updated Splits(T) keeps the set of split activities of a process definition.
Updated Criticals(i→k), ∀i, k∈T, denotes the set of critical activities of paths i→k.

Principle
Here i and s are parameters of the procedure. The procedure returns a set of ancestors of split activity i relevant to start activity s, i.e. PreSplits(i \mid s).
Temporary sets RestActivities, $V$ are used.

Procedure ($i, s$)
Step 1: PreSplits($i \mid s$) $\leftarrow \emptyset$;
Step 2: if $i = s$, stop (return PreSplits($i \mid s$));
Step 3: $V \leftarrow \{s\} \cup$ Criticals($s \rightarrow i$);
Step 4: RestActivities $\leftarrow$ Splits($T$) $\setminus \{i\}$;
Step 5: if RestActivities = $\emptyset$, stop (return PreSplits($i \mid s$));
Step 6: remove an element, say split activity $p$, from set RestActivities;
Step 7: if $p \notin V$, go to Step 5;
Step 8: PreSplits($i \mid s$) $\leftarrow$ PreSplits($i \mid s$) $\cup \{p\}$; go to Step 5.

5.2 Join Activity

For a given start activity $s$ ($s \in \text{Sources}(T)$), activity $j$ is a join activity of split activity $i$ ($i \in \text{Splits}(T)$), denoted by $j \in \text{Joins}(i \mid s)$, with the following recursive definition:

1. Paths($s \rightarrow i$) $\neq \emptyset$, or $i = s$;
2. Criticals($i \rightarrow j$) $= \emptyset$; and
3. if $(i, j) \in ^pT(j)$, exists at least one $q$, otherwise exist at least two different $q$, $(q, j) \in ^pT(j)$, $q \neq i$, with
   
   Paths($i \rightarrow q$) $\neq \emptyset$, and $j \notin \text{Criticals}(i \rightarrow q)$;
   
   and $\partial p \in \text{PreSplits}(i \mid s)$, if
   
   $p \in \text{Criticals}(i \rightarrow q) \cup \{q\}$,
   
   then
   
   $\forall m \in \text{Joins}(p \mid s) \cap \text{Reaches}(j)$, with
   
   $m \notin \text{Criticals}(i \rightarrow q) \cup \{q\}$.

For a given start activity $s$, activity $j$ is a potential join activity of split activity $i$, if

1. Paths($s \rightarrow i$) $\neq \emptyset$, or $i = s$;
2. Criticals($i \rightarrow j$) $= \emptyset$; and
3. if $(i, j) \in ^pT(j)$, exists at least one $q$, otherwise exist at least two different $q$, $(q, j) \in ^pT(j)$, $q \neq i$, with
   
   Paths($i \rightarrow q$) $\neq \emptyset$, and $j \notin \text{Criticals}(i \rightarrow q)$;

In other words, activity $j$ is a potential join activity of split activity $i$, if

1. activity $i$ is reachable from a start activity;
2. there is no critical activity on paths $i \rightarrow j$; and
3. activity \( j \) has at least two predecessors through which activity \( i \) has distinct elementary paths to activity \( j \) but not over activity \( j \).

A potential join activity meets part of the definition of a join activity. If an activity is not a potential join activity, it can not be a join activity.

If an activity is critical on paths of \( i \rightarrow j \), the activity has more opportunity than activity \( j \) being a join activity of \( i \) (see Example 5-8 in Section 5.2.1).

For a certain start activity \( s \), activity \( m \) is called ancestor join activity on an elementary path from split activity \( i \) to activity \( j \) over activity \( q \), if \( \exists p, p \in \text{PreSplits}(i \mid s) \text{ and } p \in \text{Criticals}(i \rightarrow q) \cup \{ q \} \cup \text{Criticals}(q \rightarrow j) \) with

\[
m \in \text{Joins}(p \mid s) \cap \text{Reaches}(j)
\]

Here \( j \) is a potential join activity of activity \( i \) relevant to activity \( s \), \( q \neq i \), \( q \neq j \), \( q \in \text{Activities}(i \rightarrow j) \). The set of the ancestor join activities is denoted by \( \text{PreJoins}(i \rightarrow q \rightarrow j \mid s) \).

A potential join activity \( j \) of split activity \( i \) becomes a join activity, if there is no ancestor join activity which is critical on the two distinct paths of \( i \rightarrow j \) that decide activity \( j \) being a potential join activity of split \( i \).

If activity \( j \) is a potential join activity of split activity \( i \) but not a join activity of it, parallel work items split at activity \( i \) will be joined at the ancestor join activity before they go over the ancestor of split activity \( i \) to activity \( j \) (see Example 5-13 in Section 5.2.1).

A join activity must have at least two predecessors that are reachable from a split activity. Because at a split activity, one execution thread of a process instance may split into multiple execution threads, several work items of the same process instance will flow to the join activity.

<table>
<thead>
<tr>
<th>Assumption 5-1. Synchronization at a Join Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>When a work is routed from a predecessor activity to a join activity at runtime, the join activity is not allowed to execute until there is no other work associated with the same process instance with the potential to flow to the join activity. The work must wait at a join activity with a “WaitForJoin” status. Before a join activity is executed, all work items waiting at the join activity and associated with the same process instance will be joined into one work.</td>
</tr>
</tbody>
</table>

From the assumption, at a join activity, several execution threads of a process instance will be joined to one execution thread. The assumption avoids workflow participants at the join activity to receive a work associated with the process instance several times from different predecessor activities. At runtime, the workflow engine checks regularly whether parallel work items waiting at join activities are able to be joined and to run further.
5.2.1 Examples of Join Activities

The definition of join activity used in the Espresso WFMS is somewhat complex. The following examples can help the process designers to comprehend the determination of join activities of a process definition.

Example 5-5

For the process definition in Figure 5-5:

- Sources\((T) = \{1\}\);
- Splits\((T) = \{1\}\).
- PreSplits\((1 \mid 1) = \phi\);
- Criticals\((1\rightarrow4) = \phi\);
- Criticals\((1\rightarrow2) = \phi\);
- Criticals\((1\rightarrow3) = \phi\);
- Joins\((1 \mid 1) = \{4\}\).

It is obvious that activity 4 is a join activity of split activity 1 from start activity 1. Here activity 4 meets the definition of a join activity:

1. \(1 = 1\) (activity 1 is both the start activity and split activity);
2. Criticals\((1\rightarrow4) = \phi\); and
3. \(P^T(4) = \{(2, 4), (3, 4)\}\), with
   - Paths\((1\rightarrow2) = \{(1, 2)\} \neq \phi\),
   - Paths\((1\rightarrow3) = \{(1, 3)\} \neq \phi\), and
   - activity 4 is not critical on the paths 1→2 and 1→3; and
   - because PreSplits\((1 \mid 1) = \phi\), activity 4 can not be a join activity of an ancestor of activity 1.

A process instance is created at run-time at activity 1 and an instance of activity 1 is generated. After activity 1 is completed, the work associated with the process instance flows to activity 2 and activity 3 simultaneously and then there exist two activity instances associated with the process instance. Activity 4 can be executed only after both activity 2 and activity 3 are completed and work items routed from there are joined to one instance of activity 4. After activity 4 is completed, the process instance is terminated.
Example 5-6

For the process definition in Figure 5-6,

\[
\begin{align*}
\text{Sources}(T) &= \{1\}; & \text{Splits}(T) &= \{1, 2\}. \\
\text{PreSplits}(1 \mid 1) &= \emptyset; & \text{Criticals}(1\rightarrow5) &= \emptyset, \\
\text{Criticals}(1\rightarrow2) &= \emptyset, & \text{Criticals}(1\rightarrow4) &= \emptyset, \\
\text{Criticals}(1\rightarrow3) &= \emptyset, & \text{Criticals}(1\rightarrow3) &= \emptyset; \\
\text{Joins}(1 \mid 1) &= \{5, 4\}. \\
\text{PreSplits}(2 \mid 1) &= \{1\}; & \text{Criticals}(2\rightarrow5) &= \emptyset, \\
\text{Criticals}(2\rightarrow4) &= \emptyset, & \text{Criticals}(2\rightarrow4) &= \emptyset; \\
\text{Joins}(2 \mid 1) &= \{5\}.
\end{align*}
\]

That is, activity 4 and activity 5 are join activities of split activity 1, and activity 5 is a join activity of split activity 2.

Activity 4 is not a join activity of split activity 2, because there is only one path of 2\rightarrow4.

At split activity 2, the activity instance that became parallel after split activity 1 was completed will result in two parallel activity instances of activity 4 and activity 5 after it is completed. Associated with a process instance in accordance with the process definition, at most three parallel activity instances can exist in the WfMS: one at activity 3, one at activity 4 and another at activity 5. Activity 4 can be executed only after both activity 2 and activity 3 are completed; and activity 5 can be executed only after both activity 2 and activity 4 are completed.

Example 5-7
For the process definition in Figure 5-7,

\[
\begin{align*}
\text{Splits}(T) &= \{1, 2\}. \\
\text{PreSplits}(2 \mid 1) &= \{1\}; & \text{Criticals}(2\rightarrow4) &= \emptyset; \\
\text{Joins}(2 \mid 1) &= \{5\}.
\end{align*}
\]

Figure 5-6. Multi-Split and Join

Figure 5-7. Not a Join—without Split
Sources\( (T) = \{1, 2\} \), \quad \text{Splits}(T) = \phi.

Although activity 3 has two predecessors of activity 1 and activity 2, it is not a join activity. The work items from activity 1 and activity 2 will not be joined, because they are associated with different process instances: one is created at activity 1 and another at activity 2 (see Assumption 2-2).

**Example 5-8**

For the process definition in Figure 5-8, all links are “Always” links and so

\[
\begin{align*}
\text{Sources}(T) &= \{1\}; \\
\text{Criticals}(1\rightarrow8) &= \{4\}, \\
\text{Joins}(1\mid 1) &= \{4\}, \\
\text{Criticals}(4\rightarrow8) &= \phi, \\
\text{PreSplits}(4\mid 1) &= \{1\}, \\
\text{PreJoins}(4\rightarrow6\rightarrow8\mid 1) &= \phi, \\
\text{Joins}(4\mid 1) &= \{8\}.
\end{align*}
\]

That is, activity 4 is a join activity of activity 1 and activity 8 is a join activity of activity 4.

Since Criticals\( (1\rightarrow8) = \{4\} \neq \phi \), activity 4 is critical on paths of 1\( \rightarrow8 \) and so activity 8 is not a join activity of split activity 1.

**Example 5-9**

For the process definition in Figure 5-9,

\[
\begin{align*}
\text{Sources}(T) &= \{1\}; \\
\text{Splits}(T) &= \{1\}. \\
\text{Criticals}(1\rightarrow4) &= \{3\}; \\
\text{Joins}(1\mid 1) &= \phi.
\end{align*}
\]
There is no join activity of split activity 1. Although activity 3 has two predecessors of activity 1 and activity 4 which are reachable from split activity 1, it is not a join activity of the split activity, since $3 \in \text{Criticals}(1 \rightarrow 4)$ (activity 3 is on one of the distinct paths $(1, 3, 4, 3)$).

**Example 5-10**

For the process definition in Figure 5-10,

\[
\begin{align*}
\text{Sources}(T) &= \{1\}; & \text{Splits}(T) &= \{2\}. \\
\text{Criticals}(2 \rightarrow 2) &= \emptyset, & \text{Criticals}(2 \rightarrow 3) &= \emptyset, & \text{Criticals}(2 \rightarrow 4) &= \emptyset; \\
\text{Joins}(2 | 1) &= \{2\}.
\end{align*}
\]

Note that activity 2 is a split activity and also a join activity of itself. After the work items at activity 3 and activity 4, which were split at activity 2, are completed, they will flow back to activity 2 and be joined there.

**Example 5-11**

For the process definition in Figure 5-11,

\[
\begin{align*}
\text{Sources}(T) &= \{5\}; & \text{Split}(T) &= \{1\}. \\
\text{Joins}(1 | 5) &= \{1\}.
\end{align*}
\]

Activity 1 is a join activity of itself, although activity 5, the start activity of the process definition, is on one of the two distinct paths $1 \rightarrow 1$. 
A process is created at activity 5. When activity 5 is completed, the work is routed to activity 1, and after activity 1 is completed, two instances of activity 4 and activity 5 exist. The instance of start activity 5 can be parallel. Before activity 1 is executed, synchronization of work items associated with a process instance must be considered.

Example 5-12

For the process definition in Figure 5-12,

\[
\begin{align*}
\text{Sources}(T) &= \{1\}; & \text{Splits}(T) &= \{1, 3\}.
\end{align*}
\]

\[
\begin{align*}
\text{Joints}(1 | 1) &= \{4\}, & \text{PreSplits}(3 | 1) &= \{1\};
\end{align*}
\]

\[
\begin{align*}
\text{PreJoins}(3\rightarrow1\rightarrow4 | 1) &= \{4\}, & \text{Criticals}(3\rightarrow1) &= \{2\};
\end{align*}
\]

\[
\begin{align*}
\text{Joints}(3 | 1) &= \{4\}.
\end{align*}
\]

Activity 4 is a join activity of split activity 3, although for the paths 3→1 (part of one of the two distinct paths 3→4), activity 1, an ancestor of split activity 3, belongs to Criticals(3→1) ∪ \{1\}.

A work over activity 2 can be joined at activity 4 when it flows back to split activity 1 and then to activity 4.
Example 5-13

For the process definition in Figure 5-13,

\[
\begin{align*}
\text{Sources}(T) &= \{1\}; \\
\text{Splits}(T) &= \{4, 3\}. \\
\text{Joins}(4 \mid 1) &= \{5, 8, 7\}. \\
\text{PreSplits}(3 \mid 1) &= \{4\}; \\
\text{PreJoins}(3 \rightarrow 6 \rightarrow 5 \mid 1) &= \{5, 8, 7\}, \\
\text{Criticals}(3 \rightarrow 6) &= \{4, 2, 8\}; \\
\text{Criticals}(3 \rightarrow 4) &= \{2, 8\}; \\
\text{PreJoins}(3 \rightarrow 4 \rightarrow 7 \mid 1) &= \{5, 8, 7\}, \\
\text{Joins}(3 \mid 1) &= \{8\}.
\end{align*}
\]

That is, activities 5, 7, 8 are join activities of split activity 4; activity 8 is a join activity of split activity 3.

Activity 4 is not a join activity of split activities 3, for \(\text{Criticals}(3 \rightarrow 4) \neq \emptyset\).

Activity 5 is not a join activity of split activity 3, because on the path of \(3 \rightarrow 5\) over activity 6, one of the two distinct paths of \(3 \rightarrow 5\), ancestor join activity 8 belongs to \(\text{Criticals}(3 \rightarrow 6)\). Activity 7 is not a join activity of split activity 3, because on the path \(3 \rightarrow 7\) over activity 4, one of the two distinct paths of \(3 \rightarrow 7\), ancestor join activity 8 belongs to \(\text{Criticals}(3 \rightarrow 4)\).

Parallel activity instances associated with the same process instance created at split activity 3, which is a parallel activity instance originally created at ancestor of split activity 4, are not joined at activity 5 and activity 7, but at activity 8.
5.2.2 Algorithm for Getting Ancestor Joins

This procedure is called by the algorithm for Determining Joins (see Section 5.2.3).

**Hypothesis**

$T$ denotes the activity set of a process definition. $\text{Sources}(T)$ represents the start activity set. $\text{Updated Splits}(T)$ stands for the set of all split activities of the process definition. $\text{Reaches}(k)$, $\forall k \in T$, is the set of activities which are reachable from activity $k$; $\text{Criticals}(p \rightarrow k)$, $\forall p$, $k \in T$, is the set of critical activities of paths $p \rightarrow k$.

$\text{PreSplits}(i \mid s)$ denotes a set of ancestors of split activity $i$ from start activity $s$. $\text{Joins}(p \mid s)$, $p \in \text{Splits}(T)$, represents a set of join activities of split activity $p$ relevant to start activity $s$. $\text{Joins}(p \mid s)$, $\forall p \in \text{PreSplits}(i \mid s)$, has been determined.

**Principle**

Activities $i$, $q$, $j$, and $s$ are parameters of the procedure. Here $s \in \text{Sources}(T)$ and $i \in \text{Splits}(T)$; activity $j$ is possibly a join activity of activity $i$; and activity $q$ ($q \neq i$, $q \neq j$) is reachable from activity $i$ and has a path to activity $j$. Activity $q$ is a parallel activity between split activity $i$ and activity $j$, if activity $j$ is a join activity.

Returned set $\text{PreJoins}$ keeps the set of ancestor join activities of a path of $i \rightarrow j$ over activity $q$ and relevant to start activity $s$.

Temporary sets $\text{RestSplits}$, $V$ are used.

**Procedure** $(i, q, j, s)$

Step 1: $\text{PreJoins} \leftarrow \phi$;
Step 2: $V \leftarrow \text{Criticals}(i \rightarrow q) \cup \{q\} \cup \text{Criticals}(q \rightarrow j)$;
Step 3: $\text{RestSplits} \leftarrow \text{Getting Ancestors of a Split Activity} (i, s)$ (i.e. $\text{PreSplits}(i \mid s)$);
Step 4: if $\text{RestSplits} = \phi$, stop (return $\text{PreJoins}$);
Step 5: remove an element, say split activity $p$, from set $\text{RestSplits}$;
Step 6: if $p \notin V$ (activity $p$ is not critical on paths $i \rightarrow j$ over $q$), go to Step 4;
Step 7: $\text{PreJoins} \leftarrow \text{PreJoins} \cup (\text{Joins}(p \mid s) \cap \text{Reaches}(j))$;
Step 8: go to Step 4.

5.2.3 Algorithm for Determining Joins

This procedure is called by the algorithm for Determining Workflow Control Data (see Section 5.4.2).
Hypothesis

\( T \) denotes the set of activities within a process definition. \( \text{Sources}(T) \) is the set of start activities. \( \mathcal{P}(j) \), \( j \in T \), represents the set of predecessors of activity \( j \). \( \text{Splits}(T) \) stands for the set of split activities of the process definition. Updated \( \text{Reaches}(k) \), \( \forall k \in T \), is a set of activities to which activity \( k \) is reachable. \( \text{Criticals}(i \rightarrow k) \), \( i, k \in T \), is the set of critical activities of paths \( i \rightarrow k \);

Principle

This procedure determines \( \text{Joins}(i \mid s) \), the join activities of split activity \( i \) (\( i \in \text{Splits}(T) \)) relevant to start activity \( s \) (\( s \in \text{Sources}(T) \)). Here \( i \) and \( s \) are parameters of the procedure.

Temporary set \( \text{RestActivities} \) keeps the activities that have not been determined whether to be a join activity of activity \( i \). Set \( \text{RestLinks} \) keeps not treated predecessors of current considered activity. Temporary variable \( \text{CountPaths} \) keeps current number of paths meeting the definition that an activity is join activity of split activity \( i \).

Set \( U \) is used Temporarily.

Procedure \( (i, s) \)

Step 1: \( \text{Joins}(i \mid s) \leftarrow \emptyset \);
Step 2: if \( i = s \), go to Step 4;
Step 3: if \( i \notin \text{Reaches}(s) \) (paths of \( s \rightarrow i \) do not exist), stop;
Step 4: \( \text{RestActivities} \leftarrow \text{Reaches}(i) \);
Step 5: remove an element, say activity \( j \), from \( \text{RestActivities} \);
Step 6: if \( |\mathcal{P}(j)| < 2 \) (activity \( j \) has no or only one predecessor), go to Step 19;
Step 7: \( \text{CountPaths} \leftarrow 0 \);
Step 8: \( \text{RestLinks} \leftarrow \mathcal{P}(j) \);
Step 9: remove an element, say link \( (q, j) \), from set \( \text{RestLinks} \);
Step 10: if \( q = i \) (activity \( i \) is connected to activity \( j \) by a link), go to Step 15;
Step 11: if \( q \notin \text{Reaches}(i) \) (paths \( i \rightarrow q \) do not exist), go to Step 16;
Step 12: \( U \leftarrow \text{Getting Ancestor Joins with parameters } i, q, j \text{ and } s \); \( U \leftarrow U \cup \{ j \} \);
Step 13: if \( q \notin U \) (activity \( q \) is an ancestor join activity), go to Step 16;
Step 14: if an activity in set \( U \) belongs to \( \text{Criticals}(i \rightarrow q) \), go to Step 16;
Step 15: \( \text{CountPaths} \leftarrow \text{CountPaths} + 1 \); if \( \text{CountPaths} > 1 \), go to Step 18;
Step 16: if \( \text{RestLinks} \neq \emptyset \), go back to Step 9;
Step 17: if \( \text{CountPaths} < 2 \), go to Step 19;
Step 18: \( \text{Joins}(i \mid s) \leftarrow \text{Joins}(i \mid s) \cup \{ j \} \);
Step 19: if \( \text{RestActivities} \neq \emptyset \), go back to Step 5; otherwise stop.

5.3 Parallel Activity

Activity \( k \) is a parallel activity, if \( \exists i, j \in T \setminus \{ k \} \) and \( s \in \text{Sources}(T) \), with
1. \( j \in \text{Joins}(i \mid s) \);
2. \( \text{Paths}(i \rightarrow k) \neq \emptyset, \text{Paths}(k \rightarrow j) \neq \emptyset \); and
3. \( j \notin \text{Criticals}(i \rightarrow k) \) and \( i \notin \text{Criticals}(k \rightarrow j) \).

Here imply that \( k \neq i \) and \( k \neq j \). In other words, activity \( k \) is a parallel activity between split activity \( i \) and join activity \( j \), if it is reachable from activity \( i \) without passing through activity \( j \) and meanwhile has a path to activity \( j \) without going over activity \( i \).

The set of parallel activities between split activity \( i \) and join activity \( j \) relevant to start activity \( s \) is denoted by \( \text{Parallels}(i \rightarrow j \mid s) \).

Example 5-14
For the process definition in Figure 5-14,

\[
\begin{align*}
\text{Sources}(T) & = \{1\}, & \text{Splits}(T) & = \{1, 2\}; \\
\text{Joins}(1 \mid 1) & = \{3\}, & \text{Joins}(2 \mid 1) & = \emptyset; \\
\text{Criticals}(1 \rightarrow 5) & = \{3\}, & \text{Criticals}(4 \rightarrow 3) & = \{1\}; \\
\text{Parallels}(1 \rightarrow 3 \mid 1) & = \{2, 6\}.
\end{align*}
\]

That is, activity 2 and activity 6 are parallel activities between split activity 1 and join activity 3.

Between split activity 1 and join activity 3, activity 4 is not a parallel activity because split activity 1 belongs to Criticals(4→3); activity 5 is not a parallel activity because join activity 3 belongs to Criticals(1→5).

It is logic that \( 5 \notin \text{Parallels}(1 \rightarrow 3 \mid 1) \). Parallel work items created after activity 1 is completed will be joined together at activity 3. If instance of activity 5 exists, there is no other activity instance associated with the same process instance.

It is an assumption that \( 4 \notin \text{Parallels}(1 \rightarrow 3 \mid 1) \). Although an activity is reachable from a split activity and has a path to one of its join activities, the activity is not defined as a parallel activity if the split activity is critical on the path from the activity to the join activity.
A parallel activity is reachable from a split activity and has a path to a join activity of the split activity. At run-time, if a work associated with a process instance is at a parallel activity, the work is possibly a parallel work created at a split activity and has the potential to join at the join activities of the split activity.

The set of join activities, to which the work at activity $k$ has the potential to flow, denoted by $\text{ToJoins}(k)$, is defined as

$$\{ j \mid \forall s \in \text{Sources}(T) \text{ and } \forall i \in \text{Splits}(T) \text{ with } k \in \text{Parallels}(i \rightarrow j \mid s) \}$$

That is, $\text{ToJoins}(k)$ is a set of join activities to which activity $k$ has a path and is reachable from the split activity of a relevant join activity.

If activity $k$ is a parallel activity, $\text{ToJoins}(k)$ has at least one element, i.e. $\text{ToJoins}(k) \neq \emptyset$. At run-time, if $\text{ToJoins}(k)$ is not empty, a work at activity $k$ has the potential to be joined at the join activities belonging to the set. $\text{ToJoins}(k)$, $\forall k \in T$, are important control data for the workflow engine to decide joining of work items associated with a process instance.

**Assumption 5-2. Join List (Run-time)**

If $j \in \text{ToJoins}(i)$, $i \in T$, work at activity $i$ has the potential to be joined at join activity $j$.

**Example 5-15**

For the process definition in Figure 5-5, it is known that:

$$\text{Joins}(1 \mid 1) = \{4\}.$$ 

From above definitions, we have

$$\text{Parallels}(1 \rightarrow 4 \mid 1) = \{2, 3\}; \text{ and so}$$
$$\text{ToJoins}(2) = \{4\} \text{ and}$$
$$\text{ToJoins}(3) = \{4\}.$$ 

That is, activity 2 and activity 3 are parallel activities and they have the potential to be joined at activity 4.

PM can display the results as shown in Figure 5-15. “IsJoin” beside

![Figure 5-15. To-Join List](image)
each activity icon indicates whether the activity is a join activity and “ToJoin” presents the set of join activities where work at the activity has the potential to join.

At run-time, when a work associated with a process instance comes to activity 4, it should be examined whether the work should wait for joining. If an instance of activity 2 or activity 3 associated with the same process instance exists in the WfMS, it is known through ToJoins(2) or ToJoins(3) that the work has the potential to come to activity 4. So at activity 4, synchronization for joining is needed.

5.4 Workflow Control Data

*Workflow control data* of each activity, including copy flag, join flag and to-join list, are used by the Espresso workflow engine to determine synchronization for joining and to detect and release deadlocks. They are calculated by PM and saved with activity definitions in the Process Database.

If an activity is a parallel activity, the copy flag of the activity is set to TRUE (or “1”); otherwise to FALSE (or “0”). The join flag of an activity stands for whether the activity is a join activity or not. The to-join list of an activity is the set of join activities, to which a work at the activity has the potential to flow.

As defined above, a parallel activity has a non-empty to-join list and vice versa. That is, whether copy flag is “1” or “0” can be deduced by whether the to-join list is non-empty or empty. So in the following examples, values of copy flags are not displayed.

5.4.1 Algorithm for Getting the First Split Activity

This procedure is called by the algorithm for Determining Workflow Control Data (see Section 5.4.2). Because a join activity definition concerns with ancestor join activities that are decided by ancestors of a split activity, the nearest split from a given start activity must be dealt with first.

**Hypothesis**

$T$ represents the activity set of a process definition. $\text{Sources}(T)$ denotes the set of start activities. $\text{Splits}(T)$ stands for the set of split activities. $\text{Updated Reaches}(i)$, $\forall i \in T$, is the set of activities which are reachable from activity $i$. $\text{Updated Criticals}(i \rightarrow k)$, $\forall i, k \in T$, is the set of critical activities of paths $i \rightarrow k$. 

Principle
Set $U$ and activity $s$ are parameters of the procedure. Here $U \subseteq \text{Splits}(T)$ and $s \in \text{Sources}(T)$. This procedure returns a split activity $i$, $i \in U$, if

$$i = s; \text{ or } \text{Paths}(s \rightarrow i) \neq \emptyset \text{ and } \forall p \in \text{PreSplits}(i \mid s), \text{ with } p \notin U;$$

otherwise returns 0.

Temporary set RestSplits is used.

Procedure $(U, s)$
Step 1: RestSplits $\leftarrow U$;
Step 2: if RestSplits = $\emptyset$ (required split activity can not be found), stop (return 0);
Step 3: remove an element, say split activity $i$, from set RestSplits;
Step 4: if $i = s$ (the start activity is a split activity), stop (return $s$);
Step 5: if $i \notin \text{Reaches}(s)$ (paths $s \rightarrow i$ do not exist), go to Step 2;
Step 6: if RestSplits = $\emptyset$, stop (return $i$);
Step 7: remove an element, say split activity $p$, from RestSplits;
Step 8: if $p = s$ (the start activity is a split activity), stop (return $s$);
Step 9: if $p \notin \text{Reaches}(s)$ (paths of $s \rightarrow p$ do not exist), go to Step 6;
Step 10: if $p \notin \text{Criticals}(s \rightarrow i)$ ($p$ is not an ancestor of split activity $i$), go to Step 6;
Step 11: $i \leftarrow p$; go to Step 6.

5.4.2 Algorithm for Determining Workflow Control Data

The procedure is called by the algorithm for Getting Elementary Deadlocks in the next chapter.

Hypothesis
$T$ denotes the activity set of the process definition. Copy flags, join flags, and to-join lists are respectively kept in CopyFlag($i$), JoinFlag($i$), and ToJoins($i$), $\forall i \in T$.

Sources($T$) represents the set of start activities. $T^{S}(i)$, $i \in T$, stands for the set of outgoing links of activity $i$. Split($T$) denotes the set of split activities. Reaches($i$), $i \in T$, is the set of activities that are reachable from activity $i$; and Criticals($i \rightarrow k$), $i, k \in T$, is the set of critical activities on paths $i \rightarrow k$.

Principle
This procedure determines CopyFlag($i$), JoinFlag($i$) and ToJoins($i$), $\forall i \in T$. 
Before the determination, reachable as well as critical data, and split activity set of a process definition will be updated. For each start activity, all split activities, from the nearest to the start activity till the farthest, will be individually dealt with for getting all their join activities.

A self-called sub procedure with parameters $s$, $i$, $k$ and $j$ will be called for determining parallel activities between split activity $i$ and the join activity $j$ relevant to start activity $s$. CopyFlag($n$) will be set to TRUE and activity $j$ be added to ToJoins($n$), $\forall (k, n) \in T^i(k)$ with $n \in \text{Parallels}(i \mid j \mid s)$.

Set ParallelActivities keeps the parallel activities found between split activity $i$ and join activity $j$. It is cleared before the sub procedure is called by the main procedure.

The path from a split activity built up during the determination is kept in a stack PathStack($j$), $j = 1, 2, ..., \text{StackPointer}$. When the procedure is called by the main procedure, StackPointer is assigned with 0. Except for PathStack(1), which is a split activity, the activities on the path are parallel activities from the split activity to the join activity.

Temporary sets $V$, RestStarts, RestJoins, set RestLinks, $U$ are used here.

**Main Procedure**

Step 1: call Determining Reachability and Criticality (Reaches($i$), $i \in T$, as well as Criticals($i \rightarrow k$), $\forall k \in \text{Reaches}(i)$, will be determined)

Step 2: ToJoins($i$) $\leftarrow \phi$, CopyFlag($i$) $\leftarrow$ FALSE, JoinFlag($i$) $\leftarrow$ FALSE, $\forall i \in T$;

Step 3: call Updating Split Activities (Splits($T$) is updated);

Step 4: if Splits($T$) $= \phi$, stop;

Step 5: RestStarts $\leftarrow$ Sources($T$);

Step 6: remove a start activity, say activity $s$, from set RestStarts;

Step 7: $V \leftarrow$ Splits($T$);

Step 8: $i \leftarrow$ Getting the First Split Activity with parameters $V$ and $s$;

Step 9: if $i = 0$ (no split activity reachable from activity $s$ can be found in $V$),

go to Step 19;

Step 10: remove split activity $i$ from set $V$;

Step 11: call Determining Joins with parameters $i$ and $s$ (Joins($i \mid s$) will be updated);

Step 12: RestJoins = Joins($i \mid s$);

Step 13: if RestJoins $= \phi$, go to Step 18;

Step 14: remove an element, say join activity $j$, from set RestJoins;

Step 15: JoinFlag($j$) $\leftarrow$ TRUE;

Step 16: StackPointer $\leftarrow$ 0; ParallelActivities $\leftarrow \phi$; call the sub procedure with parameters $s$, $i$, $i$ and $j$;

Step 17: go to Step 13;

Step 18: if $V \neq \phi$, go to Step 8;

Step 19: if RestStarts $\neq \phi$, go to Step 6; otherwise stop.
Sub Procedure \((s, i, k, j)\)

Step 1: \(\text{RestLinks} \leftarrow T^S(k)\);
Step 2: if \(\text{RestLinks} = \emptyset\), stop;
Step 3: remove an element, say link \((k, n)\), from set \(\text{RestLinks}\);
Step 4: if \(n = j\) (\(n\) is the join activity), go to Step 2;
Step 5: if \(n\) is in \(\text{PathStack()}\) or \(n\in\text{ParallelActivities}\) (\(n\) has been dealt with), go to Step 2;
Step 6: if \(j\notin \text{Reaches}(n)\) (paths of \(n\rightarrow j\) do not exist), go to Step 2;
Step 7: \(U \leftarrow \{i, j\}\);
Step 8: if \(n\in U\) (activity \(n\) is either split activity \(i\) or join activity \(j\)), go to Step 2;
Step 9: if an activity in set \(U\) belongs to \(\text{Criticals}(n\rightarrow j)\) (split activity \(i\) or join activity \(j\) is critical on paths \(n\rightarrow j)\), go to Step 2;
Step 10: \(\text{CopyFlag}(n) \leftarrow \text{TRUE}; \text{ToJoins}(n) \leftarrow \text{ToJoins}(n) \cup \{j\};\) \(\text{ParallelActivities} \leftarrow \text{ParallelActivities} \cup \{n\}\);
Step 11: \(\text{StackPointer} \leftarrow \text{StackPointer} + 1; \text{PathStack}(\text{StackPointer}) \leftarrow n;\)
Step 12: call the sub procedure self with parameters \(s, i, n\) and \(j\);
Step 13: \(\text{StackPointer} \leftarrow \text{StackPointer} - 1;\) go to Step 2.

Example 5-16

![Workflow Control Data](image)

Figure 5-16. Workflow Control Data

For the process definition in Figure 5-16, activity 4 is specified as a start activity. \(\text{CopyFlag}(i), \text{JoinFlag}(i)\) and \(\text{ToJoins}(i), i\in T\), are determined by the algorithm and the results are displayed beside each activity icon. That is,

\[
\begin{align*}
\text{ToJoins}(2) &= \emptyset, & \text{ToJoins}(3) &= \{5, 7, 8\}, \\
\text{ToJoins}(4) &= \emptyset, & \text{ToJoins}(5) &= \{8\}, \\
\text{ToJoins}(6) &= \{5, 8\}, & \text{ToJoins}(7) &= \{8\} & \text{and} \\
\text{ToJoins}(8) &= \emptyset.
\end{align*}
\]
Activities 2, 4 and 8 are not parallel activities; Activities 5, 7, and 8 are join activities of the process definition.

Now we change the process definition in Figure 5-16 by specifying merely activity 3 a start activity as shown in Figure 5-17 and get the following results:

- $\text{ToJoins}(2) = \{5, 7\}$
- $\text{ToJoins}(3) = \{5, 7, 8\}$
- $\text{ToJoins}(4) = \{5, 7\}$
- $\text{ToJoins}(5) = \{7, 8\}$
- $\text{ToJoins}(6) = \{5, 7, 8\}$
- $\text{ToJoins}(7) = \{5, 8\}$
- $\text{ToJoins}(8) = \{5, 7\}$

Consequently all activities in the process definition are parallel activities.

From this example we know that purely different specifications of start activities of a process definition can result quite different effects of joining opportunities.

### 5.5 Conclusion

If after execution of activity $i$, one execution thread of a process instance is split into multiple concurrent execution threads, activity $i$ is a split activity, denoted by $i \in \text{Splits}(T)$. Whether an activity is a split activity or not is determined by the set of outgoing links of the activity.

If at activity $j$, it can happen that multiple concurrent execution threads split activity $i$, which is reachable from start activity $s$, are joined into one
execution thread, activity \( j \) is a join activity of activity \( i \) and is denoted by \( j \in \text{Joins}(i \mid s) \).

If split activity \( i \) is not reachable from start activity \( s \) without \( p \) over split activity \( p \), activity \( p \) is called an ancestor of split activity \( i \), denoted by \( p \in \text{PreSplits}(i \mid s) \).

Join activity \( m \) of split activity \( p \) related to start activity \( s \) is an ancestor join activity on a path of \( i \to q \to j \), denoted by \( m \in \text{PreJoins}(i \to q \to j \mid s) \), if join activity \( m \) is reachable from activity \( j \), and split activity \( p \), which is an ancestor of split activity \( i \) related to start activity \( s \), is identical with activity \( q \) or is critical on paths \( i \to q \) or \( q \to j \). Ancestor join activities are used to determine join activities of a process definition.

Workflow control data are determined by the network as well as start activities of a process definition. They indicate whether an activity is a parallel activity as well as a join activity, and what is the set of join activities an activity is reachable. The workflow control data are used by the workflow engine for controlling the flowing of work items associated with the process definition. Figure 5-18 illustrates how the workflow control data are determined.

---

**Figure 5-18. Determining Workflow Control Data**
6 DEADLOCK

6.1 Deadlock Opportunity

According to Assumption 5-1 and Assumption 5-2, if a process definition has more than one join activity, deadlock may happen at run-time between the join activities. Because the work at a join activity can not be executed if at other activities there are work items associated with the same process instance and have the potential to flow to the join activity, parallel work items at different join activities may wait for one another and the process can not run further.

Example 6-1

![Diagram](image)

**Figure 6-1. Deadlock**

For the process definition in Figure 6-1, activity 2 and activity 3 are join activities of split activity 1. A process instance in accordance with the process definition is created at activity 1. After activity 1 is completed, the work associated with the process instance is split into two parallel work items and they are sent to activity 2 and activity 3 separately and simultaneously. When a work arrives at activity 2, it can not be executed while another work associated with the same process instance is at activity 3 and has the potential to come to activity 2. But meanwhile, the work arriving at activity 3 is waiting also for the work at activity 2 flowing to activity 3 for joining. So the parallel work items at activity 2 and activity 3 are waiting for each other and can not be executed. Thus the process instance can not run further and deadlock happens.

Two or more join activities can be involved in a deadlock. There are \( m \) \((m > 1)\) different join activities \( j_k, j_k \in T, k = 1, 2, \ldots, m, \) involved in a deadlock, if

\[
\begin{align*}
    j_m & \in \text{ToJoins}(j_1), \\
    j_k & \in \text{ToJoins}(j_{k+1}), \ k = 1, 2, \ldots, m-1
\end{align*}
\]
That is, there is a deadlock opportunity associated with the process definition: one work at join activity \( j_k \) is waiting for the work at join activity \( j_{k+1} \), \( (k = 1, 2, \ldots, m - 1) \) and one at activity \( j_m \) for that at activity \( j_1 \).

A deadlock opportunity that can be found according to the above definition is called an *elementary deadlock*.

### 6.2 Join Priority and Deadlock Release

*Priority* is “the order of importance in which requests, entries, and jobs will be handled or processed.” [Weinberg, 1980, p. 316] Join priorities assigned by a process designer will be utilized for releasing deadlocks.

**Assumption 6-1. Deadlock Release**

To release a deadlock at run-time, the workflow engine will let the parallel work at the join activity, which has the lowest value of join priority of all join activities involved in the deadlock, not wait for joining anymore and allocate the work to the worklists of workflow participants for execution.

*Join priority*, which orders joining preferences when a deadlock happens, is assigned to each activity within a process definition in PM. Value 0, as a default value of join priority, stands for the lowest joining opportunity of a join activity. That is, when a deadlock is detected among join activities, the join activity with the smallest value of join priority will be executed at first without joining. Join priorities of all join activities involved in a deadlock must be different from one another. Thus if an activity is involved in several deadlocks, these deadlocks will be grouped together for assignment of different join priorities among activities in the group.

In the following examples, *Priority*(\( i \)), \( i \in T \), refers to join priority of activity \( i \).

**Example 6-2**

To the process definition in Figure 6-1 of Example 6-1, join priority of activity 2 and activity 3 are displayed by PM in dialog box “Check Joining Priorities” as shown in Figure 6-2 and can be modified. In the activity list, the first activity has priority 1, the second 2, and so on. Join priority definition of join activities in a group is independent on that in another group.

Suppose that Priority(2) = 1 and Priority(3) = 2 for the process definition in Figure 6-1. At run-time, when a work waits at activity 2 for joining with one from activity 3, and at the same time, a work associated with the same process instance waits at activity 3 for joining with the work from activity 2 (deadlock happens), the priorities will be used. Because activity 2 has a lower value of join priority than that of activity 3, the work at activity 2 will be executed first.
3, the work at activity 2 will not wait for joining anymore and can be executed at once and thus the deadlock is released. After activity 2 is

![Figure 6-2. Join Priority Specification](image)

completed, the work is sent to activity 3 and is joined there.

### 6.2.1 Algorithm for Getting Elementary Deadlocks

This procedure called by the algorithm for Grouping Deadlocks (see Section 6.2.2) is used at build-time to find all elementary deadlock opportunities in a process definition.

**Hypothesis**

$T$ denotes the activity set of a process definition. JoinFlag($i$) stands for whether activity $i$ is a join activity and ToJoins($i$) keeps the set of join activities where a work at activity $i$ has the potential to join, $\forall i \in T$.

Set Deadlocks keeps all different elementary combinations of deadlock opportunities in a process definition.

**Principle**

A set of deadlocks of a process definition will be returned.

A self-called sub procedure with parameters $i$, $k$ and $U$ is used. Activity $i$ and activity $k$ are join activities. Set $U$ has pairs of join activities, say activity $q$ and activity $r$, denoted by $q \# r$, with $r \in$ ToJoins($q$). Current path of join activities built up during searching is kept in stack PathStack($n$), $n = 1, 2, \ldots$, StackPointer. StackPointer is assigned with 0, before the procedure is called by
the main procedure. Join activity $j, j \in \text{ToJoins}(i)$, must not be included in the searching path, if $i \# j \notin U$. It is already known that $\text{PathStack}(1) \in \text{ToJoins}(k)$. So as a result, the found path of join activities combines a deadlock opportunity and will be added to set Deadlocks.

Temporary sets $U, \text{RestActivities}$ are used here.

**Main Procedure**

Step 1: call Determining Workflow Control Data (determine $\text{JoinFlag}(i)$ and $\text{ToJoins}(i), \forall i \in T$);

Step 2: $\text{Deadlocks} \leftarrow \emptyset$;

Step 3: $U \leftarrow \emptyset$;

Step 4: $\text{RestActivities} \leftarrow T$;

Step 5: if $\text{RestActivities} = \emptyset$, go to Step 9;

Step 6: remove an element, say activity $i$, from set $\text{RestActivities}$;

Step 7: if $\text{JoinFlag}(i) = \text{TRUE}$ and $\text{ToJoins}(i) \neq \emptyset$ (activity $i$ is a join activity and may be involved in deadlocks), let $U \leftarrow U \cup \{i \# j\}, \forall j \in \text{ToJoins}(i)$;

Step 8: go to Step 5;

Step 9: if $|U| \leq 1$ (none or one pair of join activities can not combine deadlock opportunity), stop (return set Deadlocks);

Step 10: remove an element, say join activity pair $i \# j$, from set $U$;

Step 11: StackPointer $\leftarrow 0$; call the sub procedure with parameters $j, i$ and $U$;

Step 12: go to Step 9.

**Sub Procedure ($i, k, U$)**

Step 1: StackPointer $\leftarrow$ StackPointer + 1, PathStack(StackPointer) $\leftarrow i$;

Step 2: if $i = k$ (cycle of join activities exists), add sequence of join activities on the searching path (i.e. PathStack(1), PathStack(2), ...,

 $\textbf{PathStack(\text{StackPointer})}$) to set Deadlocks, and go to Step 9;

Step 3: $\text{RestActivities} \leftarrow \text{ToJoins}(i)$;

Step 4: if $\text{RestActivities} = \emptyset$, go to Step 9;

Step 5: remove an element, say join activity $j$, from set $\text{RestActivities}$;

Step 6: if $i \# j \notin U$, go to Step 4;

Step 7: if $j = \text{PathStack}(n)$ with $n \in [1, \text{StackPointer}]$ (join activity $j$ is on current searching path), go to Step 4;

Step 8: call the sub procedure self with parameters $j, k$ and $U$; go to Step 4;

Step 9: StackPointer $\leftarrow$ StackPointer − 1; stop.
6.2.2 Algorithm for Grouping Deadlocks

Elementary combinations of deadlock opportunities will be grouped for specifying join priorities and for reporting verification results of a process definition.

**Hypothesis**
A deadlock is a set of join activities that have opportunity to combine a deadlock at run-time.

**Principle**
This procedure returns a set of grouped deadlocks kept in set GroupDeadlocks. Temporary sets RestDeadlocks, CurrentGroup, and NoRelations are used. Set RestDeadlocks keeps the deadlocks that have not been grouped. Set NoRelations contains deadlocks that have been removed from RestDeadlocks but have not been grouped in set CurrentGroup. Variable ToAddGroup means whether to initiate set CurrentGroup with the first element in set RestDeadlocks.

**Procedure**
Step 1: GroupDeadlocks $\leftarrow \emptyset$;
Step 2: RestDeadlocks $\leftarrow$ Getting Elementary Deadlocks;
Step 3: ToAddGroup $\leftarrow$ TRUE;
Step 4: if RestDeadlocks = $\emptyset$, stop (return GroupDeadlocks);
Step 5: if ToAddGroup = FALSE (not to create a new group), go to Step 8;
Step 6: remove an element, say deadlock CurDeadlock, from RestDeadlocks;
Step 7: CurrentGroup $\leftarrow$ CurDeadlock;
Step 8: ToAddGroup $\leftarrow$ TRUE;
Step 9: NoRelations $\leftarrow$ $\emptyset$;
Step 10: if RestDeadlocks = $\emptyset$, go to Step 15;
Step 11: remove an element, say deadlock CurDeadlock, from set RestDeadlocks;
Step 12: if CurrentGroup $\cap$ CurDeadlock $\neq \emptyset$ (at least one join activity in deadlock CurDeadlock appears in set CurrentGroup too), go to Step 14;
Step 13: NoRelations $\leftarrow$ NoRelations $\cup$ CurDeadlock; go to Step 10;
Step 14: CurrentGroup $\leftarrow$ CurrentGroup $\cup$ CurDeadlock; ToAddGroup $\leftarrow$ FALSE; go to Step 10.
Step 15: add CurrentGroup in set GroupDeadlocks;
Step 16: RestDeadlocks $\leftarrow$ NoRelations; go to Step 4.
Example 6-3

For the process definition in Figure 6-3, Sources(T) = {1} and Splits(T) = {1, 2, 3, 5}; Joins(1 | 1) = {1, 2, 3, 4, 5, 6}, Joins(2 | 1) = {1, 4, 5, 6}, Joins(3 | 1) = {1, 4, 5, 6}, and Joins(5 | 1) = {6}.

The algorithm for Getting Elementary Deadlocks finds the following elementary deadlocks:

4→1→ (4),
4→5→1→ (4),
5→1→ (5),
5→4→1→ (5),
3→2→ (3), and
5→4→ (5).
They are combined by the algorithm for Grouping Deadlocks into two deadlock groups as \( \{2, 3\} \) and \( \{1, 4, 5\} \) for specification of join priorities in the groups (see square-marked activity icons in Figure 6-3).

### 6.2.3 Example: Deadlock Release

In the process definition of Figure 6-4, link (6, 3) and link (6, 7) are **Exclusive**. \( \mathcal{T}(T) = \{1\} \). \( \text{Splits}(T) = \{1\} \). \( \text{Joins}(1 \mid 1) = \{3, 4, 6\} \).

\( \text{Sinks}(T) = \{7\} \).

![Figure 6-4. Deadlock Release](image)

The algorithm for Getting Elementary Deadlocks will find the following elementary deadlocks:

- \( 4 \rightarrow 3 \rightarrow (4) \),
- \( 4 \rightarrow 6 \rightarrow 3 \rightarrow (4) \),
- \( 6 \rightarrow 3 \rightarrow (6) \),
- \( 6 \rightarrow 4 \rightarrow 3 \rightarrow (6) \), and
- \( 6 \rightarrow 4 \rightarrow (6) \).

Before specifications of join priorities, activities 3, 4, 6 are combined in one deadlock group. Join priorities of activities 3, 4 and 6 must be different.

The following examples explain how different specifications of join priorities influence the activity execution threads of the process instances in accordance with the process definitions with the same activity network. The protocol in a table presents an execution thread of a process instance from the start activity to the last activity. The start activity has no previous activity. If an activity has multiple previous activities, parallel work items coming from the previous activities are joined there.
6.2.3.1 Join Priority Order 1

Suppose that the order of join priorities in the deadlock group is specified as 3<4<6, or

\[ \text{Priority}(3) = 1, \text{Priority}(4) = 2, \text{and Priority}(6) = 3. \]

Process instances generated at run-time can be the execution threads like those shown in Figure 6-5. After activity 1 is completed, the work is split into three pieces that are sent to activities 3, 4 and 6 separately. Now a deadlock happens. According to the join priority specification, activity 3 can be executed without joining. After activity 3 is completed, the work is sent to activity 4 and is joined to the work from activity 1. At present there is another deadlock between activity 4 and activity 6. According to the join priority specification, activity 4 can be executed without joining. After it is completed, the work is sent to activity 6 and is joined with the work from activity 1. Afterwards, there is no more parallel work for the process instance.

6.2.3.2 Join Priority Order 2

Suppose that the order of join priorities in the deadlock group is 3<6<4, or

\[ \text{Priority}(3) = 1, \text{Priority}(6) = 2, \text{and Priority}(4) = 3. \]
The execution threads of a process instance can be one of those shown in Figure 6-6.

![Figure 6-6. Deadlock Release: 3<6<4](image)

When three parallel work items are waiting at join activities 3, 4 and 6, activity 3 can be executed without joining and then the work is sent to activity 4 after it is completed at activity 3. As it is joined at activity 4 with the work from activity 1, activity 6 can be executed without joining. After activity 6 is completed, a workflow participant of activity 6 decides routing the work along either link “back” or link “to end”. If link “back” is selected by the workflow participant, the work is sent back to activity 3 and then to activity 4, where two parallel work items associated with the same process instance are joined and no more parallel work items exist; If link “to end” is selected by the workflow participant, the two parallel work items at activities 4 and 7 have no opportunity to join and will complete separately at end activity 7 (one in execution thread of 1→6→7 and the other in one of 1→3→4→6→7, 1→3→4→6→3→4→6→7, and so on).

### 6.2.3.3 Join Priority Order 3

Suppose that the order of join priorities in the deadlock group is specified with 4<3<6, or

Priority(4) = 1, Priority(3) = 2, and Priority(6) = 3.
The execution threads of a process instance can be those as shown in Figure 6-7.

![Figure 6-7. Deadlock Release: 4<3<6](image)

When three parallel work items are waiting at join activities 3, 4 and 6, activity 4 can be executed without joining and then the work is sent from activity 4 to activity 6 and is joined with the work waiting there. Now activity 3 can be executed without joining with parallel work at activity 6. After it is completed the parallel work is sent to activity 4 and then to activity 6. At activity 6, it is joined with the parallel work waiting there. Now there is no parallel work associated with the same process instance anymore.

### 6.2.3.4 Join Priority Order 4

Similarly we can specify priorities in the deadlock group with the order as 4<6<3. The execution threads of a process instance can be those as shown in Figure 6-8. Note that, if a workflow participant of activity 6 chooses link “to end”, no join within the process instance will happen and the process instance will terminate at activity 7 two times: one with execution thread of 1→4→6→7 and another with 1→3→4→6→7.
6 DEADLOCK

6.2.3.5 Join Priority Order 5

If the join priority is specified with the order 6<3<4, the execution threads of a process instance may be the same as those shown in Figure 6-9. Note that, if the workflow participant of activity 6 choose link “to-end”, a process instance will terminate at activity 7 two times.

Figure 6-9. Deadlock Release: 6<3<4
6.2.3.6 Join Priority Order 6

Finally, if the join priority is specified with the order 6<4<3, the execution threads of a process instance may be the same as those shown in Figure 6-10.

![Figure 6-10. Deadlock Release: 6<4<3](image)

6.3 Algorithms for Handling Deadlocks

6.3.1 Algorithm for Getting One Deadlock

This algorithm can be used by the workflow engine of the Espresso WfMS. During simulation, it is called by the algorithm for Releasing Deadlock (see Section 6.3.4).

**Hypothesis**

$T$ denotes the set of activities within a process definition. ToJoins($i$), $i \in T$, represents a set of activities to which a work at activity $i$ has the potential to flow.

Set $U$, the parameter of the procedure, includes a set of join activities with non-empty ToJoins() (i.e. $\forall j \in U$, ToJoins($j$) $\neq \emptyset$). At each activity in set $U$, there is a work associated with the same process instance. Work items associated with one process instance at some of the join activities in set $U$ may have involved in a deadlock at run-time.
Principle
Set \( U \), \( |U| > 1 \) and \( U \subseteq T \), is a parameter of the main procedure. This procedure will return a deadlock combined with join activities in set \( U \).

A self-called sub procedure with parameters \( i, k \) and \( V \) is used here to return a deadlock opportunity. Activity \( i \) and activity \( k \) are join activities. Set \( V \) has pairs of join activities, say activity \( q \) and activity \( r \), denoted by \( q \# r \), with \( r \in \text{ToJoins}(q) \). Current path of join activities built up during searching is kept in stack \( \text{PathStack}(n), n = 1, 2, \ldots, \text{StackPointer} \). \( \text{StackPointer} \) is assigned with 0, before the procedure is called by the main procedure. Join activity \( j \), \( j \in \text{ToJoins}(i) \), can not be included in the searching path if \( i \# j \notin V \). It is already known that \( \text{PathStack}(1) \in \text{ToJoins}(k) \). So as a result, the found path of join activities combines a deadlock opportunity and will be returned.

Temporary sets \( \text{RestActivities} \) and \( \text{CurDeadlock} \) are used in the procedure.

Main Procedure \((U)\)
\[ \begin{align*}
\text{Step 1:} & \quad V \leftarrow \phi; \\
\text{Step 2:} & \quad \text{RestActivities} \leftarrow U; \\
\text{Step 3:} & \quad \text{if RestActivities} = \phi, \text{go to Step 7}; \\
\text{Step 4:} & \quad \text{remove an element, say join activity} \ i, \text{from set RestActivities}; \\
\text{Step 5:} & \quad \forall j \in \text{ToJoins}(i) \cap U, \text{let} \\
& \quad V \leftarrow V \cup \{i \# j\}; \\
\text{Step 6:} & \quad \text{go to Step 3}; \\
\text{Step 7:} & \quad \text{if} \ |V| \leq 1 \text{(join activity pairs in set} \ V \text{can not combine deadlock), stop (return} \ \phi); \\
\text{Step 8:} & \quad \text{remove an element, say join activity pair} \ i \# j, \text{from set} \ V; \\
\text{Step 9:} & \quad \text{StackPointer} \leftarrow 0; \ \text{CurDeadlock} \leftarrow \text{the sub procedure with parameters} \ j, \ i \text{and} \ V; \\
\text{Step 10:} & \quad \text{if} \ \text{CurDeadlock} \neq \phi, \text{stop (return} \ \text{CurDeadlock);} \\
\text{Step 11:} & \quad \text{go to Step 7}.
\end{align*} \]

Sub Procedure \((i, k, V)\)
\[ \begin{align*}
\text{Step 1:} & \quad \text{StackPointer} \leftarrow \text{StackPointer} + 1, \ \text{PathStack}(\text{StackPointer}) \leftarrow i; \\
\text{Step 2:} & \quad \text{if} \ i = k \ (\text{cycle of join activities exists}), \text{stop (return sequence of join activities on the searching path:} \ \text{PathStack}(1), \ \text{PathStack}(2), \ldots, \ \text{PathStack}(\text{StackPointer})); \\
\text{Step 3:} & \quad \text{RestActivities} \leftarrow \text{ToJoins}(i); \\
\text{Step 4:} & \quad \text{if RestActivities} = \phi, \text{go to Step 10}; \\
\text{Step 5:} & \quad \text{remove an element, say join activity} \ j, \text{from set RestActivities}; \\
\text{Step 6:} & \quad \text{if} \ i \# j \notin V, \text{go to Step 4}; \\
\text{Step 7:} & \quad \text{if} \ j = \text{PathStack}(n) \text{with} \ n \in [1, \text{StackPointer}] \text{(activity} \ j \text{is on current searching path), go to Step 4}; \\
\text{Step 8:} & \quad \text{CurDeadlock} \leftarrow \text{the sub procedure self with parameters} \ j, \ k \text{and} \ V;
\end{align*} \]
Step 9: if CurDeadlock = φ, go to Step 4, otherwise stop (return CurDeadlock);
Step 10: StackPointer ← StackPointer – 1; stop (return φ).

6.3.2 Algorithm for Routing Work to Activity

Status “WaitForJoin” set in this procedure will be used in the algorithm for Getting Parallel Waiting Work Items (see Section 6.3.3).

Hypothesis
$T$ denotes the activity set of a process definition. $\text{CopyFlag}(i)$ and $\text{JoinFlag}(i)$, $i \in T$, represent respectively whether activity $i$ is a parallel activity and a join activity. $\text{ToJoins}(i)$ stands for a set of join activities where a work at activity $i$ has the potential to be joined.

Suppose that from the nomination of a work in a WfMS, it can be identified, with which process instance and process definition the work is associated.

Principle
$\text{WorkID}$ and $i$ are parameters of the procedure. This procedure will be called when routing work $\text{WorkID}$ to activity $i$. Status of work $\text{WorkID}$ will be set to “WaitForJoin” if it should wait at activity $i$ for joining and therefore can not be executed.

Temporary sets $\text{ParallelInstances}$ and $U$ are used.

Procedure $(\text{WorkID}, i)$

Step 1: if JoinFlag$(i) = \text{FALSE}$ (activity $i$ is not a join activity), go to Step 10;
Step 2: put all other work items associated with the same process instance as that of work $\text{WorkID}$ in set $\text{ParallelInstances}$;
Step 3: If $\text{ParallelInstances} = \phi$ (there is no parallel work that has the potential to come to activity $i$), go to Step 10;
Step 4: remove an element, say work $\text{CurWork}$, from set $\text{ParallelInstances}$, and suppose that it is at activity $k$;
Step 5: if $k = i$ (work $\text{CurWork}$ arrived at activity $i$ and is waiting at the same activity as work $\text{WorkID}$), go to Step 11;
Step 6: if CopyFlag$(k) = \text{FALSE}$ (activity $k$ is not a parallel activity and so work $\text{CurWork}$ can not be joined at activity $i$), go to Step 3;
Step 7: if $i \notin \text{ToJoins}(k)$ (it is not possible that work $\text{CurWork}$ flows from activity $k$ to activity $i$), go to Step 3;
Step 8: set status of work $\text{WorkID}$ to “WaitForJoin”;
Step 9: call Detecting Deadlocks; stop;
Step 10: (work $\text{WorkID}$ does not have status “WaitForJoin” and can be sent to the workflow participants who will perform activity $i$) stop;
Step 11: merge work WorkID to work CurWork; clear status of work CurWork from “WaitForJoin”; call the procedure self with parameters CurWork and \( i \); stop.

**6.3.3 Algorithm for Getting Parallel Waiting Work Items**

This algorithm can be used by the workflow engine. During simulation, it is called by the algorithm for Detecting Deadlocks (see Section 6.3.5).

**Hypothesis**
Suppose that from the nomination of a work in a WfMS, it can be identified, with which process instance and process definition it is associated.

**Principle**
WorkID is a parameter of the procedure. It is known that a work with identical name WorkID has “WaitForJoin” status that is set in the algorithm for Routing Work to Activity. Suppose work WorkID is waiting at activity \( i \). If all parallel work items of work WorkID have “WaitForJoin” status and some of them are not waiting at activity \( i \), there is a deadlock and the procedure returns the parallel waiting work items; otherwise returns an empty set.

WaitInstances keeps the set that will be returned. Temporary set ParallelInstances is used.

**Procedure (WorkID)**
Step 1: put all work items associated with the same process instance as work WorkID in set ParallelInstances;
Step 2: if \(|\text{ParallelInstances}| = 0\) (work WorkID is not parallel anymore), stop (return work WorkID);
Step 3: WaitInstances \( \leftarrow \emptyset \);
Step 4: remove an element, say work CurWork, from set ParallelInstances, and suppose that it is at or is flowing to activity \( k \);
Step 5: if \( k = i \) (parallel work CurWork is coming to the same activity as the given work WorkID), stop (return \( \emptyset \));
Step 6: if work CurWork does not have status “WaitForJoin”, stop (return \( \emptyset \));
Step 7: WaitInstances \( \leftarrow \text{WaitInstances} \cup \{\text{CurWork}\} \);
Step 8: If ParallelInstances \( \neq \emptyset \), go to Step 4;
Step 9: WaitInstances \( \leftarrow \text{WaitInstances} \cup \{\text{WorkID}\} \); stop (return WaitInstances).
6.3.4 Algorithm for Releasing a Deadlock

This algorithm can be used by the workflow engine. During simulation, it is called by the algorithm for Detecting Deadlocks (see Section 6.3.5).

**Hypothesis**
Set $U$, a parameter of the procedure, contains parallel work items (associated with the same process instance) with status “WaitForJoin”.

**Principle**
Set $U$ is a parameter of the procedure. The procedure tries to release a deadlock combined by parallel work items in set $U$.

Temporary sets RestWaits, $V$ and CurDeadlock will be used here. Set $V$ keeps join activities where parallel work items of current work wait and which may combine a deadlock.

Procedure ($U$)
Step 1: RestWaits $\leftarrow U$; $V \leftarrow \phi$;
Step 2: if RestWaits $= \phi$, go to Step 5;
Step 3: remove a work from set RestWaits and suppose that it is waiting at join activity $j$;
Step 4: $V \leftarrow V \cup \{j\}$; go to Step 2;
Step 5: if $|V| = 1$, let CurDeadlock $\leftarrow V$;
Otherwise, let CurDeadlock $\leftarrow$ Getting One Deadlock ($V$);
Step 6: if CurDeadlock $= \phi$ (no deadlock exists within parallel work items in set $U$), stop;
Step 7: get an activity from set CurDeadlock, say activity $j$, which has the lowest join priority;
Step 8: clear “WaitForJoin” Status of the parallel work in set $U$ which is waiting at activity $j$; allocate the work to the worklist of each workflow participant for execution; stop.

6.3.5 Algorithm for Detecting Deadlocks

This algorithm can be called automatically by the workflow engine in a WfMS after a scheduled period of time. During simulation, it is called by the algorithm for Routing Work to Activity or is called when a parallel work associated with a process instance is terminated or stopped.
Hypothesis
From the name of a work in a WfMS, it can be identified, with which process instance as well as process definition the work is associated.

Principle
This procedure tries to release deadlocks combined by parallel work items with status “WaitForJoin”.
Temporary sets RestWaits and $U$ will be used here.

Procedure
Step 1: put all work items which have status “WaitForJoin” in set RestWaits;
Step 2: if RestWaits = $\emptyset$, stop;
Step 3: remove an element, say work CurWork, from set RestWaits;
Step 4: $U \leftarrow$ Getting Parallel Waiting Work Items (CurWork); if $U = \emptyset$ (work CurWork is not involved in any deadlock), go to Step 2;
Step 5: call Releasing a Deadlock with parameter $U$;
Step 6: RestWaits $\leftarrow$ RestWaits $\setminus U$;
Step 7: go to Step 2.

6.4 Conclusion
If there are multiple join activities within a process definition, work items associated with the same process instance may wait for one another for joining at run-time, and thus a deadlock happens.
Join priorities are utilized to release a deadlock at run-time. They are defined by the process designer to the join activities that are potentially involved in a deadlock. The specification of join priorities affects the execution thread of a process instance.
Figure 6-11 presents how deadlocks are handled upon workflow control data and join activities.
Figure 6-11. Detect and Release Deadlocks
7 SUMMARY

7.1 Definitions and Algorithms

In this part, the symbols listed in appendix “Symbols in Definitions and Algorithms” have been used for the process definition, verification and the runtime workflow engine.

The definitions and algorithms discussed in this part are summarized in Figure 7-1. Almost all of them have been implemented in PM. The area of Process Definition in Figure 7-1 has the user interfaces where a designer models a process definition and specifies join priorities if there are deadlock opportunities in the process definition. The Verification area contains the
interfaces for output of verification results of the process definition to the designer. The algorithms in the area of Workflow engine / Simulation are used by simulation and can also be used by the workflow engine.

7.2 Constraints in Process Modeling and Enactment

Except for Assumptions 2-1, 2-2, 2-3, 4-1, 4-2, 5-1, 5-2 and 6-1, there are almost no other constraints for process modeling and process execution at run-time. So it is quite comfortable and flexible to use PM to model various activity networks of process definitions.

The network of a process definition can be designed through laying down activities and connecting them. But it is not allowed to connect two activities in the same direction multiple times. An infinite cycle of activities is also not allowed so that any process instance created in accordance with the process definition in a WfMS can terminate.

For creating process instances at run-time, start activities of a process definition should be specified and every activity should be reachable from a start activity.

For the assumptions regarding synchronization of parallel work items at join activities, the algorithms for determining join activities and to-join lists are very important. For a certain process definition with parallel activities, the designer should ensure where parallel work items will be joined and whether this meets the real world requirements.

If deadlocks have the potential to happen, join priority can be specified by the designer for releasing a deadlock.

Animation experiments and the process protocol reports, generated during animating or simulating a process definition (see Chapter 13), can help a process designer to foresee where parallel work items will be joined and how deadlocks will be released.

7.3 Further Work*

The following points are not considered in this work. For a more complex and sophisticated process-modeling tool, they could be implemented.

- Searching redundant links.
  If there is a link, say \((i, j)\), connecting a split activity to a join activity of the split activity, the link is redundant if there is another strong path from the split activity to the join activity. A strong path of \(i \rightarrow j\) ensures that a work at activity \(i\) flows certainly to activity \(j\). So work flowing along link \((i, j)\) should wait for joining at activity \(j\) before it can be executed. If link \((i, j)\) is
not used for informing the workflow participants of activity \( j \) preparing execution of the activity, it makes nonsense and therefore is redundant.

A strong path only consists of “Always” and “Exclusive Choice” links.

- **Synchronization definition.**
  In Figure 7-2, the deadlock opportunity between activity 4 and activity 5 may be used to define synchronization of execution of both activities. In this case, join priorities of the two activities should be able to be specified with the same value and thus by deadlock releasing, both the activities can run further without need of waiting for joining.

![Figure 7-2. Deadlock for Synchronization](image)

- **Arrange activities automatically.**
  Use the concepts and algorithms of partition and tearing (see [Steward, 1981]) to
  1. Group activities in a cycle for automatic simplification of the network of a process definition.
  2. Ungroup the activity groups that have been grouped for this purpose.
  3. Arrange activities on the process map, for example, arrange a specific start activity at most top-left position.

- **Suggest link combination for breaking infinite cycles** (see [Steward, 1981]). For example, a non-“Always” link that appears most times in different cycles is suggested as a feedback link.
• Determine shortest and longest process duration.

Shortest duration of a process instance from creation to termination will be determined by supposing that no feedback happens. Slack may also be used for determining the earliest and latest time duration of each activity.

Longest duration is determined according to

1. a specified maximum number of feedback of an activity in an escalation definition, or
2. specification of the latest termination time of an activity (larger than the earliest start time of the activity and the latest completion time of all predecessors plus the execution time of the activity).

Because due date can be specified by the creator of a process instance at run-time, determination of the least duration of a process instance in accordance with a process definition is necessary.

Here is the definition and the algorithms offered by [Steward, 1981].

• The earliest start and completion times are the earliest times when an activity can be started or completed such that all the required predecessor activities are completed as early as possible:

1. The earliest start time of a start activity is the creation time of the process instance.
2. The earliest completion time of an activity is the earliest start time plus the execution time of the activity.
3. The earliest start time of an activity that has predecessors is the largest of the earliest completion times of its predecessors.
4. The earliest termination time of a process instance is the largest of the earliest completion times of end activities.

• The latest start and completion times are the latest times when an activity can be started or completed without delay of the process instance:

1. The latest completion time of an end activity is taken as the required termination time for the process instance.
2. The latest start time is its completion time minus its execution time.
3. The latest completion time of an activity that has successors is the smallest of the latest start times of the activities that succeed it.

• The slack is the length of time an activity can be delayed from its earliest time without delaying the process instance. The slack for an activity is its latest start time minus its earliest start time, or, equivalently, its latest completion time minus its earliest completion time.
It should be noticed that the above algorithms do not consider the shortages of human and synchronous material resources defined for execution of an activity.
8 INTRODUCTION OF SIMULATION

8.1 Simulation

All systems are governed by certain relationships that describe the interaction between different factors or attributes, represented by variables. These relationships may represent physical laws, economic principles, statistical correlations, etc. Some variables that act upon the system but are not acted on by the system are causes of the relationships or system input variables, and others are effects of the relationships. So the relationships are in general terms referred to as cause-and-effect relationships. See [Gottfried, 1984, p. 5].

The following definitions of simulation have been given by Gottfried and Naylor according to the meaning and the technology aspect respectively.

“Simulation is an activity whereby one can draw conclusions about the behavior of a given system by studying the behavior of a corresponding model whose cause-and-effect relationships are the same as (or similar to) those of the original system.” [Gottfried, 1984, p. 8]

“Simulation is a numerical technique for conducting experiments on a digital computer, which involves certain types of mathematical and logical models that describe the behavior of a business or economic system (or some component thereof) over extended periods of real time.” [Naylor, 1966, p. 3]

Simulation is not the best way to study and analyze a system. “Simulation is simultaneously one of the easiest to understand and one of the most misunderstood of the management science techniques. Mathematicians, computer scientists, and engineers sometimes denigrate simulation because it is purportedly not based on elegant, theoretical, general models as is, for example, linear programming. Managers and business people sometimes have been told that simulation is a panacea that always solves any problem; these individuals may then be dismayed to find that simulation may be more expensive, more time consuming, and less accurate than they were lead to believe.” [Solomon, 1983, p. ix]

8.1.1 Disadvantages of Simulation

A simulation study of a real world system is based upon the established model. “The purpose of a simulation is to produce numbers whose interpretation leads to an improved understanding of the system being simulated. Unfortunately circumstances can easily arise in a simulation study that lead to a misinterpretation of the data and consequently to a misunderstanding of the system. These circumstances include the following:
1. poorly chosen pseudorandom number generators,
2. inappropriate approximate random variate generation techniques,
3. input parameter misspecification,
4. programming errors,
5. model misspecification,
6. data collection errors in simulation,
7. poor choice of descriptors (parameters) to estimate,
8. peculiarities of the estimation method,
9. numerical calculation errors,
10. influence of initial conditions on data,
11. influence of final conditions on data and on estimation method, and
12. misuse of estimates.” [Fishman, 1973, p. 262]

A model including any of above-mentioned circumstances can not represent a real world system, and so no correct conclusion can be drawn from the simulation study upon such a model.

“Simulation typically is nothing more or less than the technique of performing sampling experiments on the model of the system. The experiments are done on the model rather than on the real system itself only because the latter would be too inconvenient, expensive, and time consuming.” [Hillier, 1974, p. 621] But a simulation model may become expensive in terms of manpower and computer time. “Development of a computer simulation of a complex system requires the services of a variety of skilled professional personnel: statisticians, operations research analysts, subject-matter specialists, computer programmers, and systems analysts. For most effective use, these personnel must be grouped into teams.” [Maisel, 1972, p. 35] In addition, hidden critical assumptions may cause the model to diverge from reality; model parameters may be difficult to initialize (these may require extensive time in collection, analysis, and interpretation).

The disadvantages of a simulation study discourage a system analyst to utilize it for analysis of a system.

8.1.2 Why Simulation

“Computer simulation becomes a legitimate research tool when known analytical methods cannot supply a solution to a problem.” [Fishman, 1973, p. 18]

One of the main strengths of this approach of formulating and solving mathematical models that represent real systems is that it abstracts the essence of the problem and reveals its underlying structure, thereby providing insight into the cause-and-effect relationships within the system. Therefore, if it is possible to construct a mathematical model that is both a reasonable idealization of the problem and amenable to solution, this analytical approach usually is superior to simulation. However, many problems are so complex that
they can not be solved analytically. Thus, even though it tends to be relatively expensive procedure, simulation often provides the only practical approach to a problem. See [Hillier, 1974, p. 620].

In addition, simulation brings other advantages. The following principal reasons for choosing computer simulation are summarized by [Naylor, 1966, pp. 8-9].

1. Simulation makes it possible to study and experiment with the complex system, such as a firm, an industry, an economy, or some subsystem of one of these.

2. Through simulation one can study the effects of certain informational, organizational, and environmental changes on the operation of a system, such as a process definition.

3. Detailed behavior observation of the system being simulated may lead to a better understanding of the system and to suggestions for improving.

4. Simulation can be used as a pedagogical device for teaching basic skills in theoretical analysis, statistical analysis, and decision making.

5. The knowledge and experience obtained in designing a simulation study frequently suggests changes in the system being simulated. The effects of these suggested changes can be tested via simulation before implementing them on the actual system.

6. Simulation of complex systems can yield valuable insight into how variables, which represent system attributes, interact.

7. Simulation can be used to experiment with new situations about which little or no information is prepared for what may happen.

8. Simulation can be used to try out new system operating policies, before running the risk of experimenting on the real system.

9. In addition to information about expected values and moments, simulation offers the sequence of events that cause a stochastic system change.

10. A dynamic system can be simulated in either real time, compressed time, or expanded time.

11. Simulation can be used to help foresee bottlenecks and other problems that may arise in the operation of the system, especially when new components are introduced into a system.

12. To simulate a system, analysts are forced into an appreciation and understanding of all facets of the system, with the result that conclusions are less apt to be biased by particular inclinations and less apt to be unworkable within the system framework.

“In the course of an experiment it is occasionally desirable to stop the experiment and review the result to date. This means that all phenomena associated with the experiment will have to retain their current states until resumption of the experiment begins. In field experiments a complete halt of all processes is seldom possible. Computer simulation, however, offers this
convenience, provided that the termination part of the program contains instructions for recording all relevant states. When the experiment resumes, the terminal states become the initial states so that no loss of continuity occurs.” [Fishman, 1973, p. 17]

8.2 Basic Simulation Terminology

8.2.1 System Categorization

The concepts of the system discussed in this section are shown in Figure 8-1.

“A system is a collection of regularly interacting or interdependent components (such as machines, people, information, and communications), acting as a unit in carrying out an implicitly or explicitly defined mission.” [Maisel, 1972, p. 8]

“The objective in studying one or several phenomena in terms of a system are to learn how change in state occurs, to predict change, and to control change. Most studies combine these objectives with varying emphasis. One particular combination of these objectives, called the evaluation of alternatives, concerns the relationship between the input to and the output from a system.” “Input refers to stimuli external to a system that induces changes in the system state. Output refers to measures of these state changes.” [Fishman, 1973, p. 6]

“From a systems analysis standpoint there are two general types of system—deterministic and stochastic (or probabilistic). In a deterministic system, the individual system components always behave in a well-defined, predictable manner. Stochastic systems, on the other hand, involve the occurrence of random events. Such systems are encountered when analyzing many realistic problems, such as games of chance, sales forecasts, financial acquisitions, equipment maintenance, inventory control, networks, and situations involving queues (waiting lines).” [Gottfried, 1984, pp. 2-3]
“The output of a deterministic system can be predicated completely if the input and the initial state of the system are known.” “However, a stochastic system in a given state may respond to a given input with any one among a range or distribution of outputs.” “It is impossible to predict the particular output of a single observation of the system.” [Maisel, 1972, pp. 13-14]

“Stochastic systems can be further categorized as being either static or dynamic. In a static system the occurrence of random events is independent of the passage of time. Such systems are relatively easy to simulate. On the other hand, the random events in a dynamic system must occur sequentially with respect to time (for example, a customer cannot enter a service area until the previous customer has departed). Thus, dynamic systems are relatively complicated and, therefore, more difficult to simulate.” [Gottfried, 1984, p. 3]

Since workflow is the automation of a business process and WfMS is a system that defines, creates and manages the execution of business processes, a WfMS is obviously a dynamic stochastic system.

### 8.2.2 Simulation Model

“We define a model as the body of information about a system gathered for the purpose of studying the system.” [Gordon, 1978, p. 6]

Models used in system studies have been classified in many ways. “According to one dimension of classification, the most common types of models are physical, schematic, mathematical, and heuristic. A physical model may be an identical replication of the real system, such as an experimental aircraft or a fashion model, or it may be scaled down, such as the wind tunnel version of the same aircraft or a doll analogous to the fashion model. A schematic model is a pictorial representation of the system, such as a blueprint or a graph. A mathematical model consists of expressions containing variables, constants, and operators which describe the process of interest. A heuristic model is a collection of descriptors and decision rules, usually computer-based, which is not limited by the physical, diagrammatic, or mathematical bounds of the other types of models. While mathematical models may be implemented on a computer, they are restricted to purely mathematical operations such as arithmetic, algebra, and calculus. Heuristic models may be programmed to search data sets and perform logical comparisons as well as mathematics.” [Solomon, 1983, p. 5]

A model for simulating on a digital computer is comprised of a set of cause-and-effect relationships within the system to describe the behavior of an actual system. The model is used to evaluate both the state of the system and some particular quantity that is used for generating system performance. By specifying different sets of conditions and evaluating the model repeatedly for each case, we can see how the system behaves in response to changes in various system input variables.
“Models which permit the decision maker to observe the status of a system over time as well as at particular points in time are often called simulation models.” [Solomon, 1983, p. 6]

A simulation study is based on a simulation model of a real world system. The model for simulation study is different from that for analytical study. “Rather than directly describing the overall behavior of the system, the simulation model describes the operation of the system in terms of individual events of the individual components of the system. In particular, the system is divided into elements whose behavior can be predicted, at least in terms of probability distributions, for each of the various possible states of the system and its inputs. The interrelationships between the elements also are built into the model.” [Hillier, 1974, pp. 620-621]

“It should be emphasized that, like any operations research model, the simulation model needs not be a completely realistic representation of the real system. In fact, it appears that most simulation models err on the side of being overly realistic rather than overly idealized. With the former approach, the model easily degenerates into a mass of trivia and meandering details, so that a great deal of programming and computer time is required to obtain a small amount of information. Furthermore, failing to strip away trivial factors to get down to the core of the system may obscure the significance of those results that are obtained.” [Hillier, 1974, p. 625]

The Espresso simulation model is implemented in PM. It is a visual tool for simulating performance or behavior of automated business processes created and routed in the WfMS based on the process definitions. The input of the model is specified in various simulation settings (see chapters 11, 12 and 13) and the output is categorized in diverse simulation reports (see Chapter 13).

### 8.2.2.1 Next-Event Approach

“An important feature of any simulation is the manner in which the model represents and advances simulated time.” [Hartley, 1975, p. 171]

An event denotes change in the state of a system, such as creating a new process instance or completing an activity in a WfMS. Some events are conditional on the occurrence of another event. For example, adding a new activity to the worklist of a workflow participant causes him to become busy when he is idle; a completion of an activity causes the workflow participant to become idle when no other activities are waiting.

“In a discrete event system change occurs when an event occurs. Since the states of entities remain constant between events, there is no need to account for this inactivity time in our modeling. Accordingly all modern computer simulation programming languages use the next event approach to time advance. After all state changes have been made at the time corresponding to a particular event, simulated time is advanced to the time of the next event, where required state changes are again made. Then simulated time is again
advanced to the time of the next event, and the process is repeated. In this way a simulation is able to skip over the inactive time whose passage in the real world we are forced to endure.” [Fishman, 1973, p. 23]

From [Lewis, 1979, p. 104], three things are needed to perform the discrete-event simulation.

- **Clock**: keeps track of the simulated time, updated by the model algorithm, initially with a zero value;
- **Eventlist**: a list of events scheduled to be processed by the procedures of the model algorithm;
- **Model algorithm**: the procedures for manipulating the eventlist and updating the clock.

According to cause-and-effect relationships, various events will be scheduled during simulation. There is an event-processing algorithm for each kind of scheduled event. The eventlist is usually sorted to the nearest scheduled time for picking out the next occurring event.

PM uses next-event approach as the model algorithm. The unit of the clock is determined by the analyst before a simulation run. It can be either minute, hour, day, or week. When unifying a time from a larger unit (such as week or day) to that of a smaller unit (such as hour or minute), the standard weekly working hours are used and weekends (Saturday and Sunday) as well as holidays are excluded. The standard weekly working hours, for example 8 hours a day and 5 days a week, is the normal working time of the people in an organization.

### 8.2.2.2 Algorithm of Next-Event Approach

Next-event approach is used in PM as the model algorithm.

**Hypothesis**
SimulationClock represents the clock. The eventlist is sorted to the scheduled time when an event will occur.

**Principle**
At the beginning of a simulation run, the clock and statistic variables (see Section 8.2.3.4) are cleared to value zero; the eventlist and state variables (see Section 8.2.3.3) are initiated to the system states when a simulation run begins. During simulation, the clock is advanced to the occurrence time of the first events kept in the eventlist, and thus the event occurs. During processing of an occurring event, state variables as well as statistic variables of the simulation model will be altered, and newly scheduled events may be inserted in the eventlist. A simulation run will be terminated when the clock exceeds a
specified simulation end time, there is no event in the eventlist anymore, or the analyst interrupts the run of simulation.

**Procedure**

Step 1: get input data from simulation settings;

Step 2: clear all statistic variables; initiate all state variables;

Step 3: \( \text{SimulationClock} \leftarrow 0; \)

Step 4: initiate the eventlist and ensure it is not empty (e.g. for simulating a WfMS, schedule for each start activity of all simulated process definitions a process creation event);

Step 5: remove the first (nearest occurring) event from the eventlist and suppose it occurs at time \( t; \)

Step 6: \( \text{SimulationClock} \leftarrow t; \) if the value of SimulationClock exceeds the specified simulation end time, go to Step 10;

Step 7: if the event is to terminate simulation run, go to Step 10;

Step 8: call a corresponding algorithm to process the event (some state variables and statistic variables change, and new scheduled events may be added to the eventlist that is sorted to scheduled event occurring times);

Step 9: if the eventlist is not empty, go to Step 5;

Step 10: generate simulation reports according to the statistic variables; stop.

### 8.2.3 Variables

System factors and attributes are represented by variables in a simulation model. Variables used in a simulation model can be classified as decision variables, system parameters, state variables, statistic variables, or performance criteria. Figure 8-2 illustrates the role of the simulation model in providing a description of system behavior.

![Simulation Model and Variables](image)

**Figure 8-2. Simulation Model and Variables**
Decision variables and system parameters are input variables of the model and are assumed to have been predetermined via different simulation settings before a simulation run in PM. Some of these input variables are, for all practical purposes, deterministic, whereas others are stochastic.

System performance criteria are statistical output variables of a model. Some values of state variables observed during simulation can also be important information for analyzing system performance.

### 8.2.3.1 Decision Variable

In most simulation studies there are certain variables or parameters that can be manipulated or controlled by decision-makers (or policy makers) of the system at the beginning of a simulation run, independent of any other considerations. These variables are known as decision variables. The values chosen for the decision variables will affect the state of the system or system performance. So the state variables and statistic variables are dependent on the decision variables.

For example, decision variables determined by the system analyst before simulating an Espresso WfMS can be:

- the process definitions implemented in the WfMS;
- environment design and specification of the WfMS, consisting of relevant Application Databases as well as configured Organization Databases and/or Notes Organization Directories;
- life period specification of a process definition;
- structural definition of a process definition (that is, the sets of activities, links and start activities);
- human and material resources assigned to each activity;
- resource availability (definitions of organizational roles in the Organization Databases and the Notes Organization Directories, including specification of weekly work hours of a resource to a process definition);
- queuing rule (e.g. first-in-first-out) for work items in a workflow participant’s worklist.

Values of decision variables can be changed by a decision-maker. The change of the decision variables affects the state of the system, and hence the manner in which the system behaves. Therefore a given set of values for the decision variables is referred to as an operating policy. Whenever a different value is assigned to one or more of the decision variables, a new operating policy is created. A change in the model (for example, an alteration of one or more cause-and-effect relationships) also results in a different operating policy.
“Our overall objective in carrying out a simulation study is to determine the best possible operating policy, relative to some particular measure of system performance. In practice, however, the amount of time and effort required to find the ‘best possible’ policy may be excessive. Therefore, in reality, we often settle for an operating policy that is reasonably satisfactory, even though it may not be the very best that is attainable.” [Gottfried, 1984, p. 6]

8.2.3.2 System Parameter

System parameters are similar to decision variables in the sense that they are input variables of a model and their values can be specified a priori. System parameters usually represent physical constants, design parameters, constants of proportionality, marketing factors, etc., over which the decision maker has little or no control. Therefore the values of the system parameters may not change from one problem situation to another, whereas the decision variables will take on different values.

All the system parameters chosen for a simulation model influence a state variable or a statistic variable. So the state variables and statistic variables are dependent upon system parameters.

Here are some system parameters appearing in a WfMS:

- fix costs for execution of an activity,
- resource hourly costs,
- intercreation time of process instances (that is, the time interval between two conjunctive created process instances) at each start activity of a process definition,
- probability for routing along a “Multiple Choice” link,
- routing distribution among outgoing “Exclusive Choice” links of an activity,
- variate distributions of a variable defined in a conditional link,
- routing time of a work from one activity to another,
- time distribution functions for activity execution, material occupation and work routing,
- public holidays of an organization, and
- activity execution distribution among members of a team.

8.2.3.3 State Variable

The state of a system can be thought of as the totality of all relevant system characteristics. It varies as time elapses. State variables describe the state of a
system or one of its components. Events occurring in a system make values of the state variables change.

In order to obtain values for the state variables, it will be necessary to carry out a number of calculations which may be rather complicated for certain types of problems. See [Gottfried, 1984, p. 4].

State variables during a particular time period are determined by values of input variables at that time and values of state variables themselves in preceding periods according to the system’s operating characteristics. If escalation is implemented in a WfMS, the state variables may be dependent upon some statistic variables. The cause-and-effect relationships are changed when an escalation procedure is invoked.

A state variable in a simulation model mostly affects a statistic variable that will be used to create performance criteria when a simulation experiment terminates.

The state variables in a WfMS include:

- the clock for tracking simulation time,
- status of a workflow participant (i.e. busy or idle),
- number of running process instances in a WfMS,
- state of a running process instance (executed by a workflow participant, waiting for joining, or waiting for resource availability),
- work items in the worklist of a workflow participant, etc.

### 8.2.3.4 Statistic Variable

Statistic variables are accumulated data from state variables as well as input variables updated during simulation according to the statistic calculation. Statistic variables are used for computing system performance criteria or for escalation of a WfMS.

The following three classified statistic variables are applied in the Espresso simulation model:

- Count of a particular kind of event. For example, total number of created process instances in accordance with a process definition. This kind of statistic variable can be directly used as a system performance criterion.
- Accumulation of a state variable. For example, total activity execution time. It can be used for computing an average data, here the average execution time for an activity, through dividing the sum by the total number of completed activities.
- Integrated value of a state variable to time. It is prepared for calculating time average statistical data, such as average queue length during a simulation run.
The time average statistical data is calculated through dividing the integrated value of the variable by the integrating period of time. For example, the plot in Figure 8-3 illustrates the values of the state variable of a queue length, denoted by $Q(t)$, from time $t_0$ to $t_{18}$. Where $t_i$, $i = 1, 2, \ldots, 18$, is the point in time when event $i$ occurs and makes the queue length change.

The average queue length between time $t_0$ and $t_{18}$ are calculated by (see [Tijms, 1986, p. 15])

$$\frac{1}{t_{18} - t_0} \int_{t_0}^{t_{18}} Q(t) dt$$

The integrated value in the formula is in fact the shaded area in Figure 8-3. Because the value of a state variable is changed only when an event occurs, so the integrated value of a state variable can be calculated with (see [Gordon, 1978, pp. 189-191])

$$\int_{t_0}^{t_{18}} Q(t) dt = \sum_{i=1}^{18} Q(t_{i-}) (t_i - t_{i-1})$$

Here $Q(t_{i-})$ is the queue length between $t_{i-1}$ and $t_i$ (that is, the queue length just before an event occurs at time $t_i$). For this example, $Q(t_{3-}) = 2$. So the integrated value of the queue length to time can be accumulated with value

$$Q(t_{i-}) (t_i - t_{i-1})$$

at each time $t_i$ when an event changing the queue length occurs.
8.2.3.5 System Performance Criterion

Some specific criteria are required as a measure of system performance. They are called the system performance criteria. System performance criteria are calculated from statistic variables that are determined from state variables and input variables. They are used for decision making or policy choosing.

The system performance criteria created by the Espresso simulation model are contained in various simulation statistic reports. They are

- total number of completed process instances,
- average number of running process instances in the system,
- average duration of a process instance,
- average duration of an activity,
- resource costs and fixed costs,
- utilization of a resource,
- average queue length waiting for resource availability,
- average join-waiting time, and so on.

8.3 Conclusion

Although simulation has some disadvantages, it is sometimes the only feasible tool for analyzing and experimenting a complex system, such as a WfMS that is usually a dynamic stochastic system.

A simulation model is a programmed heuristic model that permits the decision-maker to observe the state of a system over time as well as at particular points in time. Next-Event approach is used to advance the clock in the simulation model of a discrete event system.

Decision variables and system parameters are input data of a simulation model; state variables and performance criteria are output data of a simulation model. An operating policy is a given set of values for the decision variables designed for a simulation run. The overall objective in carrying out a simulation study is to determine the best possible operating policy, relative to some particular criteria of system performance.

Whether a simulation model is suitable to simulate a real world system is determined by the following factors:

1. well-chosen pseudorandom number generators,
2. appropriate approximate random variate generation techniques,
3. no programming errors,
4. no model misspecification of cause-and-effect relationships,
5. no numerical calculation errors,
6. easy usage of the simulation model,
7. acceptable expense of computer time for simulation, and
8. comprehensive simulation outputs usable for decision-making.

It should be noticed that the simulation results only make sense if the input sufficiently represents the important behavioral characteristics of the real system. For this purpose, statisticians, operations research analysts, subject-matter specialists, and systems analysts, should work together to collect data and determine the input to the simulation model. They are also required for analyzing the output from simulation runs for making decisions.
9 STATISTIC THEORY AND METHODS

9.1 Random Variable

“The term random variable is used to mean a real-valued function defined over a sample space associated with the outcome of a conceptual chance experiment. A particular outcome of an experiment, i.e. a numerical or sample value of a random variable, is called a random variate.” [Naylor, 1966, p. 43]

Whether a random variable is discrete or continuous depends “on the type of value assigned to the outcomes. If a random variable assumes a discrete number (finite or countably infinite) of values, it is called a discrete random variable. Otherwise it is called a continuous random variable.” [Graybeal, 1980, p. 23]

Capital letters will be used to denote random variables and lower case letters random variates. For example, $F(x)$, the cumulative distribution function for a random variable $X$, denotes the probability that $X$ is less than or equal to the particular variate $x$. In a similar manner, $f(x)$ represents the value of the probability density function of the continuous random variable $X$ when $X = x$, and $p_i$ represents the probability of a discrete random variable $X$ when $X = x_i$. See [Naylor, 1966, p. 43]. That is,

$$F(x) = \int_{-\infty}^{x} f(x) dx$$

or

$$F(x) = \sum_{i=1}^{k} p_i, \quad \text{for } x \leq x_k$$

Mean, denoted by $\mu$, is used to indicate the central tendency or location of the distribution. It is given by

$$\int_{-\infty}^{\infty} x f(x) dx$$

or

$$\sum_{i=1}^{\infty} x_i p_i$$

Variance, denoted by $\sigma^2$, is used as a measure of dispersion. The square root of the variance, that is $\sigma$, is called the standard deviation. Variance is given by
or
\[ \sum_{i=1}^{\infty} (x_i - \mu)^2 p_i \]

See [Fishman, 1973] and [Graybeal, 1980] for further discussions about properties of cumulative distribution functions and density functions.
Random variables are classified according to their probability density functions.

9.2 Distribution Functions

Some distribution functions that can be applied in the simulation study of an Espresso WfMS are introduced here. The uniform, normal, exponential, gamma or empirical distribution can be specified to generate time and other random variates in the simulation model. The \( t \), \( \chi^2 \) or \( F \)-distribution can be used to estimate distribution of a random variable and evaluate the simulation results.

9.2.1 Uniform Distribution

The uniform distribution, denoted by \( U(a, b) \), also called a rectangular distribution, is one in which the density function is a constant in the range \([a, b]\), with \( a < b \). The density function of uniform distribution is given by

\[
f(x) = \begin{cases} 
1 / (b - a), & a \leq x \leq b \\
0, & \text{otherwise}
\end{cases}
\]

The cumulative distribution function is given by

\[
F(x) = \begin{cases} 
0, & x < a \\
\frac{x - a}{b - a}, & a \leq x \leq b \\
1, & x > b
\end{cases}
\]

The mean \( \mu \) and variance \( \sigma^2 \) for this distribution are (see [Graybeal, 1980, p. 47])
\[ \mu = (a + b)/2 \text{ and } \sigma^2 = (b - a)^2/12 \]

“This distribution is used to model truly random events. If a sequence of values is chosen at random on the interval \( a \leq x \leq b \), it has the uniform distribution.” [Graybeal, 1980, p. 47]

“The rectangular distribution is the simplest probability distribution in that the probability is uniform over the whole range of the variate. Because of its simplicity, the rectangular distribution is sometimes used as an approximation to a more complex distribution, when a detailed simulation model is not required.” [Hartley, 1975, p. 65]

An example of the uniform distribution with \( \mu = 5 \) and \( \sigma = 2 \) is plotted in Figure 9-1. The range of the distribution of this example can be obtained by

\[ \mu \pm 3\sigma \]

That is,

\[ a = 1.54 \text{ and } b = 8.46 \]

In the range \([1.54, 8.46]\), the density function has the constant value about 0.14.
During simulation of the Espresso WfMS, it is possible that $\sigma = 0$ and this results in that $a = b = \mu$. In this case the variable governed by the distribution $U(a, b)$, or $U(\mu, \mu)$, has always the value of $\mu$.

A uniform distribution with the range $(0, 1)$ is called standard uniform distribution, denoted by $U(0, 1)$. Suppose the variate $u$ of a random variable $U$ following standard uniform distribution, the cumulative distribution function is given by

$$\text{Prob}(U \leq u) = F(u) = (u - a) / (b - a) = u, \text{ for } 0 \leq u \leq 1$$

This feature of standard uniform distribution is utilized to generate variates governed by other distributions (see Section 10.4.1).

### 9.2.2 Normal Distribution

“Probably the most common continuous distribution is the normal distribution. It has been found useful in modeling most measurement phenomena, such as scores on a test, heights and weights, and errors made in manufacturing processes.” [Graybeal, 1980, p. 47]

In various realistic physical situations, there are many types of random events that are governed by the normal distribution. This distribution is characterized by a symmetric, bell-shaped probability density, given by

$$f(x) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{1}{2} \left( \frac{x - \mu}{\sigma} \right)^2}$$

where $\mu$ is the mean and $\sigma$ is the standard deviation. The normal distribution is denoted by $N(\mu, \sigma)$.

Figure 9-2 illustrates an example of the normal distribution with $\mu = 5$ and $\sigma = 2$.

A normal distribution with $\mu = 0$ and $\sigma = 1$ is called a standard normal distribution, denoted by $N(0, 1)$.

If random variables $X$ and $Z$ follow $N(\mu, \sigma)$ and $N(0, 1)$ distributions respectively, then

$$(X - \mu) / \sigma = Z$$

or

$$X = \mu + \sigma Z$$
The central limit theorem makes the normal distribution probably the most useful distribution in the simulation study. The central limit theorem states that the sum of $n$ identically distributed independent random variables $X_1, X_2, \ldots, X_n$ tends to be normally distributed with a mean $n\mu$ and variance $n\sigma^2$ as $n$ tends to infinity, where $\mu$ and $\sigma^2$ are respectively the mean and the variance of $X_i$ ($i = 1, 2, \ldots, n$). See [Graybeal, 1980, p. 49 and p. 92]. That is, statistic

$$\frac{\bar{X} - \mu}{\sigma / \sqrt{n}}$$

follows the $N(0, 1)$ distribution.

"Roughly stated, this theorem means that variables resulting from the combination of many separated effects tend to be normally distributed." [Maisel, 1972, p. 52]

9.2.3 Exponential Distribution

The exponential distribution "has been used to model ‘sudden and catastrophic’ failures such as equipment failures due to manufacturing defects and light bulbs burning out. It has also been used to characterize service times and interarrival times in queueing systems." [Graybeal, 1980, p. 49]

The probability density of an exponential distribution is
\[ f(x) = \alpha e^{-\alpha x}, \quad x \geq 0 \]

where \( \alpha \) is a positive constant. The cumulative distribution function is

\[ F(x) = 1 - e^{-\alpha x} \]

The mean \( \mu \) and variance \( \sigma^2 \) for the exponential distribution given by [Graybeal, 1980, p. 49] are

\[ \mu = 1/\alpha \quad \text{and} \quad \sigma^2 = 1/\alpha^2 \]

That is, the exponential distribution’s mean \( \mu \) and standard deviation \( \sigma \) are the same.

An example of the density function of the exponential distribution with \( \mu = 5 \) is illustrated in Figure 9-3.

![Figure 9-3. Exponential Distribution with \( \mu = 5 \)](image)

See [Gottfried, 1984, p. 85] for better comprehension of the physical significance of exponential distribution.
9.2.4 Gamma Distribution

The probability density of the *gamma distribution* is given by [Gottfried, 1984, p. 90] as

\[
f(x) = \frac{\alpha^\beta x^{(\beta-1)} e^{-\alpha x}}{(\beta - 1)!}
\]

where \(\alpha\) is a positive constant and \(\beta\) is a positive, integer-valued constant. The mean for this distribution is \(\mu = \beta/\alpha\) and the variance is \(\sigma^2 = \beta/\alpha^2 = \mu/\alpha\).

The plot of the gamma distribution with \(\mu = 5\) and \(\sigma = 2\) is given in Figure 9-4.

![Figure 9-4. Gamma Distribution with that \(\mu = 5\) and \(\sigma = 2\)](image)

This distribution is often used to represent empirical data because it can take on a variety of shapes, depending upon the mean and standard deviation.

The gamma variable \(X\) can be interpreted as the sum of \(\beta\) exponentially distributed random variables, each having an expected value of \(1/\alpha\). Thus

\[X = X_1 + X_2 + \ldots + X_\beta\]

where

\[f(x_i) = \alpha e^{-\alpha x_i}\]
When mean and standard deviation of a gamma distribution are the same, the gamma distribution has the plot of an exponential distribution.

The gamma distribution can also be defined for non-integer values of $\beta$, although physical applications of this nature are less common (see [Gottfried, 1984, p. 92]). See [Fishman, 1973, pp. 208-209] for further discussion.

### 9.2.4.1 Algorithm for Drawing Gamma Distribution

**Hypothesis**

It is known that a random variate is gamma distributed and with mean $\mu$ and standard deviation $\sigma$. This algorithm is used by PM for drawing plot of a gamma distribution chosen for a random variable (see Figure 9-4).

**Principle**

Because $\mu = \beta/\alpha$ and $\sigma^2 = \beta/\alpha^2 = \mu/\alpha$, so $\beta = \text{INT}(\mu^2/\sigma^2)$ and $\alpha = \beta/\mu$, if $\beta$ is integer-valued.

For drawing gamma distribution, $f(x)$ should be determined for every given value of $x$. Directly using the density function may cause the computer overflow. So it will be calculated via the following formula:

$$
\ln f(x) = \beta \ln \alpha + (\beta - 1) \ln x - \alpha x - \ln \prod_{i=1}^{\beta-1} i
$$

$$
= \ln \alpha - \alpha x + (\beta - 1) (\ln \alpha + \ln x) - [\ln 1 + \ln 2 + \ldots + \ln (\beta - 1)]
$$

Temporary variable $p$ is used to keep the calculation of $\ln f(x)$.

**Procedure ($\mu, \sigma$)**

Step 1: let $\beta \leftarrow \text{INT}(\mu^2/\sigma^2)$;
Step 2: if $\beta < 1$, let $\beta \leftarrow 1$;
Step 3: let $\alpha \leftarrow \beta/\mu$;
Step 4: $p \leftarrow \ln \alpha - \alpha x$;
Step 5: $i \leftarrow 1$;
Step 6: $p \leftarrow p + (\ln \alpha + \ln x) - \ln i$;
Step 7: $i \leftarrow i + 1$; if $i < \beta$, go to Step 6;
Step 8: stop and return $e^p$. 
9.2.5 Empirical Distribution

In many realistic problems the probability that an event will occur is expressed in terms of empirical, grouped data. Figure 9-5 illustrates a typical set of grouped data.

![Empirical Density Function](image)

**Figure 9-5.** Empirical Density Function

There are several adjacent subintervals, which are numbered consecutively (that is, \( j = 1, 2, \ldots, m \)). Each subinterval is represented as a rectangle whose lower and upper interval bounds are \( XL_j \) and \( XU_j \) respectively. The height of each subinterval, denoted by \( f_j \), represents the probability that the value of \( X \) for a random event will fall into the \( j \)th subinterval, that is, \( XL_j \leq X \leq XU_j \). Since \( f_j \), \( j = 1, 2, \ldots, m \), represents the probability, the sum must equal 1, that is,

\[
f_1 + f_2 + \ldots + f_m = 1
\]

Figure 9-6 shows the cumulative distribution of the distribution presented in

![Empirical Cumulative Distribution Function](image)

**Figure 9-6.** Empirical Cumulative Distribution Function

Figure 9-5. The height of each subinterval, \( Y_j \) now represents the probability that the value for a random event \( X \) does not exceed \( XU_j \), or
\[
Y_j = \text{Prob}(X \leq XU_j)
\]

A 0-1 distribution, or Bernoulli distribution called by [Lehman, 1977, p. 143], is a particular empirical distribution with two subintervals to represent whether a random event occurs or not. The probability that the event occurs is often given, such as \( p \)—the probability that the event does not occur is then \( 1 - p \).

### 9.2.6 Student’s \( t \)-Distribution*

Student's \( t \)-distribution is used primarily to test differences in means of two samples selected from normally distributed populations. The density function of this distribution given by [Maisel, 1972, p. 59] is

\[
f(x) = \frac{\Gamma\left(\frac{n + 1}{2}\right)}{\sqrt{n\pi} \Gamma\left(\frac{n}{2}\right) \left(1 + \frac{x^2}{n}\right)^{(n+1)/2}}, \text{ for } -\infty < x < \infty
\]

where \( n \) is a parameter that ordinarily takes on integer values and \( \Gamma(n) \) is the complete gamma function defined by

\[
\Gamma(n) = \int_0^\infty e^{-t} t^{n-1} \, dt
\]

and having the property

\[
\Gamma(n + 1) = n\Gamma(n), \text{ for any } n > 0
\]

and

\[
\Gamma(1/2) = \sqrt{\pi} = 1.77245385 \ldots
\]

If \( n \) is an integer, then

\[
\Gamma(n + 1) = n!
\]

The \( t \)-distribution has a mean of 0 and a variance of \( n/(n - 2) \), \( n > 2 \). The only parameter in this distribution is \( n \), and—as is the case in many distributions used for statistical test—this parameter is called the degrees of freedom.
A Student’s \( t \)-density function with \( n = 4 \) is plotted in the Figure 9-7 given by [Maisel, 1972, p. 59]. A standardized normal distribution \( N(0, 1) \) is plotted on the same coordinates to illustrate the similarity of these distributions.

![Figure 9-7. \( t \)-Distribution and \( N(0, 1) \) Distribution](image)

The \( t \)-distribution has been shown to be useful in hypothesis testing. Let \( Z \) and \( C \) be independent random variables, where \( Z \) follows a standard normal distribution and \( C \) a \( \chi^2 \) distribution with \( v \) degrees of freedom. Then \( t = Z / (C/v) \) follows a \( t \)-distribution with \( v \) degrees of freedom. See [Graybeal, 1980, p. 51].

The \( t \)-distribution arises quite often when normal distributions are sampled. If \( \bar{Y} \) denotes the sample mean, \( s \) the sample standard deviation, and \( n \) the size of the example, the statistic

\[
t = n^{1/2} (\bar{Y} - \mu) / s
\]

follows a \( t \)-distribution with \( n - 1 \) degrees of freedom. See [Graybeal, 1980, p. 51].

### 9.2.7 \( \chi^2 \) Distribution*

The \( \chi^2 \) (chi-square) distribution is used in goodness-of-fit tests and in certain other nonparametric test. Its density function is

\[
f(x) = \frac{x^{(n/2)-1} e^{-x/2}}{\Gamma(n/2)2^{n/2}}, \text{ for } 0 \leq x < \infty.
\]

Here \( n \) is the degree of freedom. For the \( \chi^2 \) distribution, the mean is \( n \), and the variance is \( 2n \).

\( \chi^2 \) density functions for several values of \( n \) are plotted in the Figure 9-8 given by [Maisel, 1972, p. 60].

This distribution arises often when the squares of standard normal distributions are combined. If \( Z_i, i = 1, 2, ..., n \), are independent standard
normal random variables, then \( Z_1^2 + Z_2^2 + \ldots + Z_n^2 \) is \( \chi^2 \) distribution with \( n \) degrees of freedom. See [Graybeal, 1980, p. 51].

\[ f(x) = \frac{1}{\Gamma(m/2)\Gamma(n/2)} \frac{m^{m/2}n^{n/2}x^{(m/2)-1}(m+nx)^{-(m+n)/2}}{x} \quad \text{for} \quad 0 \leq x < \infty \]

The mean of the \( F \)-distribution is

\[ m/(n-2) \]

and the variance is

\[ \frac{m^2(n+2)}{m(n-2)(n-4)} \]

The \( F \)-distribution is also useful in hypothesis testing. Let \( C_1 \) and \( C_2 \) be two independent \( \chi^2 \) random variables with \( v_1 \) and \( v_2 \) degrees of freedom respectively. Then \( (C_1/v_1)/(C_2/v_2) \) is distributed as an \( F \)-distribution with \( v_1 \) and \( v_2 \) degrees of freedom. See [Graybeal, 1980, p. 51].
This distribution arises when one is sampling from two normal populations. Let sample 1 consist of \( n_1 \) points from a normal population with mean \( \mu_1 \) and variance \( \sigma_1^2 \), and let \( s_1^2 \) be the sample variance. Let sample 2 consist of \( n_2 \) points from a normal population with mean \( \mu_2 \) and variance \( \sigma_2^2 \), and let \( s_2^2 \) be the sample variance. Then \( \left( s_1^2 / \sigma_1^2 \right) / \left( s_2^2 / \sigma_2^2 \right) \) is distributed according to the \( F \)-distribution with \( n_1 - 1 \) and \( n_2 - 1 \) degrees of freedom. See [Graybeal, 1980, p. 51].

### 9.3 Estimation and Hypothesis

“At times a random variable \( X \) that is being used to represent some aspect of a simulation model is known to follow a particular distribution. If this is the case the researcher’s task is greatly simplified. More often than not, however, all that is known about the distribution of a random variable is what can be gleaned from the study of a set of sample values that has been collected through observations. Some technique is then needed to characterize the behavior of the random variable.” [Graybeal, 1980, p. 55]

### 9.3.1 Mean and Standard Deviation

“Whether a random variable of interest in a simulation study is represented by an empirical distribution or is known to follow a particular distribution, the analyst encounters the problem of estimating the appropriate parameters of the distribution.” [Graybeal, 1980, pp. 59-60]

Suppose for a random variable \( X \), individually collected values with the sample size \( n \) are \( x_i \), \( i = 1, 2, \ldots, n \). For most problems, the mean value of a random variable \( \mu \) is estimated by

\[
\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i
\]

Variance \( \sigma^2 \) is determined by

\[
s^2 = \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})^2
\]

or equivalently, as
Standard deviation $\sigma$ is estimated by $s$.

Note that the equations for estimating $\sigma^2$ are often written with $(n - 1)$ rather than $n$ in the denominator. The numerical difference between these two expressions becomes negligible, if $n$ is large, as it typically is the case when carrying out a computer-based simulation study.

### 9.3.2 Distribution Function

To assume the distribution of a random variable from collected values of the variable, two general approaches have been taken. “The first is to construct an empirical distribution using least squares or some other suitable curve-fitting technique. This approach should be used when the random variable does not appear to follow any of the common distributions. The second approach is to hypothesize that the random variable follows a particular distribution and to use statistical methodology to test the validity of this hypothesis. This approach is the more common of the two and, if successful, yields a distribution function that may be expressed analytically and whose behavior in most cases is well known.” [Graybeal, 1980, p. 55]

“In order to treat a random event mathematically, it is convenient to define a frequency function, or density function, which will associate the proper probability with each possible outcome. Empirically, the probability of an outcome is measured by the relative frequency of that outcome. Therefore, an empirical density function associates relative frequencies with outcomes.” [Maisel, 1972, p. 45]

Suppose $x_i, i = 1, 2, \ldots, n$, represents the $i$th value of a random variable. Sometimes the observed (or computed) values of $x_i$’s are grouped into successive intervals. A set of relative frequencies, which represents density function of an empirical distribution, can be obtained from these grouped data as follows. Let $n_j$ be the number of values of $x_i$’s falling into the $j$th interval (where $j$ is an interval index that ranges from 1 to $m$), that is,

$$n = \sum_{j=1}^{m} n_j$$

Then the relative frequency for the $j$th interval $f_j$ is simply

$$f_j = n_j / n$$
Note that the relative frequencies are required, by definition, to sum to unity. Thus
\[
\sum_{j=1}^{m} f_j = \frac{1}{n} \sum_{j=1}^{m} n_j = 1
\]
as expected.

Finally, a cumulative distribution can be obtained from the relative frequencies as

\[
\begin{align*}
Y_1 &= f_1 \\
Y_2 &= f_1 + f_2 \\
&\vdots \\
Y_j &= f_1 + f_2 + \ldots + f_j \\
&\vdots \\
Y_m &= \sum_{j=1}^{m} f_j = 1
\end{align*}
\]

Here

\[Y_j = \text{Prob}(X \leq XU_j)\]

and \(XU_j\) is the upper bound of \(j\)th subinterval.

The relative frequencies and the cumulative distribution can, of course, be expressed either as fractions, as in the above expressions, or as percentages. In the latter case, the fractional values are simply multiplied by 100. Either form is acceptable, though there may be some bias toward the use of percentages within a business environment.

**9.4 Tests of Hypotheses**

**9.4.1 Introduction**

If it is hypothesized that a random variable \(X\) comes from some known common distribution, statistical methods can then be used to assess the validity of the hypothesis. The four possible outcomes of the hypothesis-testing procedure are (see [Maisel, 1972, pp. 68-69]):

1. the hypothesis actually is true, but the test leads to its rejection as untenable;
2. the hypothesis actually is true, and the test leads to its acceptance ;
3. the hypothesis actually is false, and the test leads to its rejection; or
4. the hypothesis actually is false, but the test leads to its acceptance as valid.

Outcomes 2 and 3 are desirable, and the specifications for the statistical test should be designed to maximize the probability of these outcomes. Outcome 1 is called a type I error, and outcome 4 a type II error.

The size of the type I error, denoted by \( \alpha \), is the probability that a valid hypothesis will be rejected. The quantity \( 1 - \alpha \) is called the level of significance of the test. Specifications commonly are designed to ensure that \( \alpha \leq 0.05 \), but this is an arbitrary value, and larger type I errors might be accepted in particular situations. See [Maisel, 1972, p. 68].

The size of the type II error, denoted by \( \beta \), is related to the degree of falsity of the hypothesis being tested. For example, suppose that in one real system, the actual difference between average execution times of one activity and another is 1 minute; in a second real system, the difference in average execution time is 20 minutes. Obviously with a particular statistical test, the false hypothesis that the average execution times are the same is more likely to be accepted in the case of the first real system than in the case of the second. Thus, when a size is specified for the type II error, the degree of difference associated with this error must also be specified. The quantity \( 1 - \beta \) is called the power of the test.

“Obviously one of the objectives in hypothesis testing is to minimize \( \alpha \) and \( \beta \), the probabilities of making an incorrect decision. Unfortunately if one probability is reduced, the other is increased. In fact, the only way to simultaneously decrease both risks is to base the decision on a sample statistic obtained from a larger sample. In most testing situations \( \alpha \) is set as some predetermined acceptable level and the decision rule is formulated to minimize \( \beta \).” [Graybeal, 1980, p. 62]

In the following sections, it will be outlined how to test a hypothesis with \( \alpha \). In a business environment, \( \alpha \) will be called a rejection probability and \( (1 - \alpha) \) a confidence level. To compute \( \beta \), one must assume the hypothesis, e.g. “\( \mu = 12 \)”, is false and another alternate hypothesis, e.g. “\( \mu = 14 \)”, is true. For a given statistical test (with fixed sample size and fixed \( \alpha \)), \( \beta \) and the power \( (1 - \beta) \) can then be calculated corresponding to various assumed levels for the extent of falsity of the hypothesis as shown in Figure 9-9 given by [Graybeal, 1980, p. 65].
9.4.2 \( t \)-Test

\( t \)-test can arise in many cases for test. Here \( t \)-test is introduced by determining the mean value of a variable upon the collected data.

If the collected \( Y_i, i = 1, 2, \ldots, n \), is normally distributed, then the statistic

\[
\frac{\bar{Y} - \mu}{s / \sqrt{n}}
\]

is known to be distributed in accordance with the \( t \)-distribution. Moreover this statistic is approximately \( t \)-distributed even if the \( Y_i \) is not normally distributed, provided they are symmetrical about the mean. See [Gottfried, 1984, p. 167]. Thus we can write

\[
-t_{n-1,1-\alpha/2} < \frac{\bar{Y} - \mu}{s / \sqrt{n-1}} < t_{n-1,1-\alpha/2}
\]

where \( t_{n-1,1-\alpha/2} \) represents a tabulated value of the \( t \)-statistic having \((n - 1)\) degrees of freedom (see Table 1 in appendices) and \( \alpha/2 \) represents either of the shaded areas shown in Figure 9-10 given by [Gottfried, 1984, p. 168]. Note that
the quantity \((1 - \alpha)\) is the corresponding confidence level, that is, the likelihood that the value obtained from equation

\[
\frac{\bar{Y} - \mu}{s} \sqrt{n - 1}
\]

will fall within the unshaded area in Figure 9-10.

It is more convenient to rewrite the equation as

\[
\bar{Y} - t_{n-1,1-\alpha/2} s / \sqrt{n - 1} < \mu < \bar{Y} + t_{n-1,1-\alpha/2} s / \sqrt{n - 1}
\]

This equation tells us that the true mean \(\mu\) falls within the interval

\[
\bar{Y} \pm t_{n-1,1-\alpha/2} s / \sqrt{n - 1}
\]

at 100(1 - \(\alpha\))% confidence level. Thus, if the level of significance \((1 - \alpha)\) is specified, an appropriate value of \(t_{n - 1, 1 - \alpha/2}\) can be obtained from the Table 1. The corresponding interval bound can then be determined.

### 9.4.3 \(N(0, 1)\) Test

If the sample size \(n\) is at least 25 or 30, the central limit theorem provides that the sample mean will be normally distributed around their population mean \(\mu\). The mean of the sample means is the population mean. The standard deviation of the distribution of sample means is equal to the population standard deviation \(\sigma\) divided by the square root of the sample size \(n\). See [Solomon, 1983, p. 229].

Thus \(N(0, 1)\) test is particularly useful for evaluating simulation results. According to the central limit theorem, the statistic

\[
\frac{\bar{Y} - \mu}{s} \sqrt{n - 1}
\]

has approximately the \(N(0, 1)\) distribution if \(n\) is large enough \((n \geq 30)\). Here \(Y_i\), \(i = 1, 2, \ldots, n\), is a collected data of a random variable.

So the true mean \(\mu\) of the random variable falls within the interval

\[
\bar{Y} \pm Z_{0.5-\alpha/2} \frac{s}{\sqrt{n - 1}}
\]

at 100(1 - \(\alpha\))% confidence level. Where \(Z_{0.5-\alpha/2}\) is a tabulated valued of the \(N(0, 1)\) distribution as shown in Table 2. For example, given \(\alpha = 0.05\), \(Z_{0.5-\alpha/2} = Z_{0.475} = 1.96\).
9.4.4 $\chi^2$ Test

Before the simulation of a WfMS, the distribution of process creation at the start activities of process definitions, for example, must be characterized. “In many cases the random variable of interest is assumed to follow a particular distribution. Of course, the results obtained by the simulation study are usually very sensitive to this assumption. Thus there must be a method by which the assumption of a particular distribution can be checked. The chi-square goodness-of-fit test has proven useful in this regard.” [Graybeal, 1980, p. 70]

The $\chi^2$ statistic is used to determine how well a set of observations can be represented by a given distribution, provided each observation falls into one of $k$ different categories. If the number of observed events $O_i$ and the expected number of events $E_i$ are known for each category, then the $\chi^2$ statistic can be determined as

$$\chi^2 = \frac{(O_1 - E_1)^2}{E_1} + \frac{(O_2 - E_2)^2}{E_2} + \ldots + \frac{(O_k - E_k)^2}{E_k}$$

with $k - 1$ degrees of freedom.

It should be noted that the $\chi^2$ distribution of statistic given by above equation is only approximate. The accuracy of the approximation increases as $E_i$ increases. Normally, it is recommended that $E_i$ exceeds 5 when using this equation. See [Gottfried, 1984, p. 37], [Maisel, 1972], [Graybeal, 1980, p. 71] and [Solomon, 1983, p. 23]

This statistic is used in conjunction with a $\chi^2$ table, as given in Table 3. The rejection probability $\alpha$ given in the top row of the table indicates the probability of incorrectly rejecting the assumed distribution as shown in Figure 9-11.

![Figure 9-11. The Critical Region of $\chi^2$ Distribution](image)

When carrying out a statistical test, the hypothesis that the observed results can be represented by the given distribution is essentially tested. The chance that this hypothesis is incorrect (that is, that the distribution is inappropriate) increases as the calculated value for $\chi^2$ increases. Hence, the likelihood of
incorrectly rejecting the assumed distribution decreases. See [Gottfried, 1984, p. 35].

In practice, the hypothesis is rejected if the calculated $\chi^2$ value exceeds the tabulated value for some reasonably small rejection probability (say $\alpha = 0.05$ or $\alpha = 0.01$), since it would be highly unlikely that the observed results would differ so greatly from the expected results if the hypothesis were valid. Furthermore, the hypothesis is usually rejected if the calculated $\chi^2$ value is smaller than the tabulated value for some fairly large rejection probability (for example, $\alpha = 0.95$ or $\alpha = 0.99$); in this case, it would be highly unlikely that the observed results would fit the given distribution so perfectly. Hence, the hypothesis is accepted if the calculated $\chi^2$ value falls within the interval that is formed by the tabulated values corresponding to the two extreme rejection probabilities. See [Gottfried, 1984, p. 35].

### 9.4.5 F-Test

The F-test is, in most cases, used to test the hypothesis that the variances of two populations are equal. The test statistic is the ratio of the larger sample variance to the smaller one. If the hypothesis is true, the true ratio of population variances must be one. See [Maisel, 1972, pp. 75-76].

Suppose, it is wished to determine whether the observed values of $s_1^2$ and $s_2^2$ obtained from two samples with sizes $n_1$ and $n_2$ respectively indicate a significant difference in the variances of the true populations. The test statistic $F$-ratio is given by

$$F = s_1^2 / s_2^2$$

with $n_1 - 1$ and $n_2 - 1$ degrees of freedom, if $s_1^2 > s_2^2$

or

$$F = s_2^2 / s_1^2$$

with $n_2 - 1$ and $n_1 - 1$ degrees of freedom, if $s_2^2 > s_1^2$

For example, if given $F = 1.20$ with 15 and 15 degree of freedom for $\alpha = 0.05$, from Table 4, a critical value for $F$-distribution is 2.40. That is, $F_{m,n,\alpha} = F_{15, 15, 0.05} = 2.40$. Because the observed value $F (=1.20)$ is smaller than the critical value ($=2.40$), it can be concluded that there is no significant difference between $s_1^2$ and $s_2^2$.

### 9.5 An Example of Estimation and Test*

A start activity of an implemented process definition is observed randomly on some days between 8/17/99 to 9/29/99 and 100 process intercreation time,
denoted respectively by $Y_i$ for $i = 1, 2, \ldots, 100$, are collected as the following (in minutes).

- 8/17/99: 40, 64, 42, 3, 19, 124, 10, 32, 73, 86, 54
- 8/27/99: 20, 28, 11, 56, 122, 9
- 8/30/99: 21, 50, 8, 13, 23, 43, 49, 50
- 8/31/99: 24, 7, 69, 104, 127, 9, 57, 66
- 9/09/99: 21, 116, 1, 39, 8, 15, 82, 13, 2
- 9/10/99: 48, 95, 53, 24, 9, 18, 18, 8, 70
- 9/13/99: 4, 85, 19, 16, 19, 10, 122, 29, 13, 30, 24, 9, 7
- 9/27/99: 143, 15, 19, 16, 13, 23, 26, 58, 19
- 9/28/99: 33, 23, 9, 10, 9, 10, 13, 85, 11, 37, 13
- 9/29/99: 46, 23, 28, 146, 34, 20

From the 100 collected values, the parameters mean $\mu$ and variance $\sigma^2$ can be estimated respectively from

$$\overline{Y} = \frac{Y_1 + Y_2 + \ldots + Y_{100}}{100} = 37.79$$
$$s^2 = \frac{(Y_1^2 + Y_2^2 + \ldots + Y_{100}^2)}{100} - \overline{Y}^2 = 36.61$$

The maximum observed value is 146. We divide the range of [0, 150) into 10 subintervals of the same size and group the 100 values into these subintervals as shown in the following table:

<table>
<thead>
<tr>
<th>Category</th>
<th>Subinterval</th>
<th>Observed Frequency</th>
<th>Relative Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[0, 15)</td>
<td>32</td>
<td>0.32</td>
</tr>
<tr>
<td>2</td>
<td>[15, 30)</td>
<td>28</td>
<td>0.28</td>
</tr>
<tr>
<td>3</td>
<td>[30, 45)</td>
<td>10</td>
<td>0.10</td>
</tr>
<tr>
<td>4</td>
<td>[45, 60)</td>
<td>10</td>
<td>0.10</td>
</tr>
<tr>
<td>5</td>
<td>[60, 75)</td>
<td>5</td>
<td>0.05</td>
</tr>
<tr>
<td>6</td>
<td>[75, 90)</td>
<td>4</td>
<td>0.04</td>
</tr>
<tr>
<td>7</td>
<td>[90, 105)</td>
<td>2</td>
<td>0.02</td>
</tr>
<tr>
<td>8</td>
<td>[105, 120)</td>
<td>2</td>
<td>0.02</td>
</tr>
<tr>
<td>9</td>
<td>[120, 135)</td>
<td>4</td>
<td>0.04</td>
</tr>
<tr>
<td>10</td>
<td>[135, 150)</td>
<td>3</td>
<td>0.03</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>100</td>
<td>1</td>
</tr>
</tbody>
</table>

The plot of the relative frequency of the intercreation time is displayed in Figure 9-12. That looks like the plot of the density function of an exponential distribution with $\mu = 40$ as shown in Figure 9-13. So it is assumed that the intercreation time of process instances at the start activity has an exponential distribution with mean 40 minutes.
First $F$-test is used to test for $\alpha = 0.05$ the hypothesis that the intercreation time follows a distribution with standard deviation $40$. Because

$$s^2 = \frac{36.61^2}{40^2} = \sigma^2$$

Thus

$$F = \frac{\sigma^2}{s^2} = \frac{40^2}{36.61^2} = 1.194,$$

with $\infty$ and $100 - 1$ degree of freedom.

From Table 4, it is known that

$$F_{\infty, 60, 0.05} = 1.39 \text{ and } F_{\infty, 120, 0.05} = 1.25$$
So

\[1.39 = F_{\infty, 60, 0.05} > F_{\infty, 100 - 1, 0.05} > F_{\infty, 120, 0.05} = 1.25\]

Because

\[F = 1.194 < 1.25 = F_{\infty, 120, 0.05} < F_{\infty, 100 - 1, 0.05}\]

or

\[F < F_{\infty, 100 - 1, 0.05}\]

thus it can be concluded that there is no significant difference between \(s^2 (= 36.61^2)\) and the estimated \(\sigma^2 (= 40^2)\). That is, the hypothesis that the intercreation time has standard deviation 40 is acceptable at confidence level 95%.

Now we have a test of the above hypothesis that the intercreation time follows the exponential distribution with mean 40 minutes (note that \(\mu = \sigma\) for an exponential distribution). The expected frequency of an exponential distribution can be calculated via

\[
\text{Prob}(0 \leq X \leq x) = F(x) = 1 - e^{-\alpha x}
\]

Here \(\alpha = 1/\mu = 1/40\). So

\[
\text{Prob}(a \leq X < b) = \text{Prob}(0 \leq X < b) - \text{Prob}(0 \leq X < a) = e^{-a/40} - e^{-b/40}
\]

Expected frequency of each category with subinterval \((a, b)\) are compared with the observed frequency in the following table:

<table>
<thead>
<tr>
<th>Category</th>
<th>Subinterval [a, b)</th>
<th>Observed Frequency O_i</th>
<th>Expected Frequency E_i ((= n (e^{-a/40} - e^{-b/40})))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[0, 15)</td>
<td>32</td>
<td>31.27 (= 100(1.0000 - 0.6873))</td>
</tr>
<tr>
<td>2</td>
<td>[15, 30)</td>
<td>28</td>
<td>21.49 (= 100(0.6873 - 0.4724))</td>
</tr>
<tr>
<td>3</td>
<td>[30, 45)</td>
<td>10</td>
<td>14.77 (= 100(0.4724 - 0.3247))</td>
</tr>
<tr>
<td>4</td>
<td>[45, 60)</td>
<td>10</td>
<td>10.16 (= 100(0.3247 - 0.2231))</td>
</tr>
<tr>
<td>5</td>
<td>[60, 75)</td>
<td>5</td>
<td>6.97 (= 100(0.2231 - 0.1534))</td>
</tr>
<tr>
<td>6</td>
<td>[75, 90)</td>
<td>4</td>
<td>4.80 (= 100(0.1534 - 0.1054))</td>
</tr>
<tr>
<td>7</td>
<td>[90, 105)</td>
<td>2</td>
<td>3.30 (= 100(0.1054 - 0.0724))</td>
</tr>
<tr>
<td>8</td>
<td>[105, 120)</td>
<td>2</td>
<td>2.26 (= 100(0.0724 - 0.0498))</td>
</tr>
<tr>
<td>9</td>
<td>[120, 135)</td>
<td>4</td>
<td>1.56 (= 100(0.0498 - 0.0342))</td>
</tr>
<tr>
<td>10</td>
<td>[135, 150)</td>
<td>3</td>
<td>1.07 (= 100(0.0342 - 0.0235))</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
Since expected frequencies in categories 6 to 10 are less than 5, we combine them together and get the table:

<table>
<thead>
<tr>
<th>Category</th>
<th>Subinterval</th>
<th>Observed Frequency $O_i$</th>
<th>Expected Frequency $E_i$</th>
<th>$(O_i - E_i)^2/E_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[0, 15)</td>
<td>32</td>
<td>31.27</td>
<td>0.0170</td>
</tr>
<tr>
<td>2</td>
<td>[15, 30)</td>
<td>28</td>
<td>21.49</td>
<td>1.9721</td>
</tr>
<tr>
<td>3</td>
<td>[30, 45)</td>
<td>10</td>
<td>14.77</td>
<td>1.5405</td>
</tr>
<tr>
<td>4</td>
<td>[45, 60)</td>
<td>10</td>
<td>10.16</td>
<td>0.0025</td>
</tr>
<tr>
<td>5</td>
<td>[60, 75)</td>
<td>5</td>
<td>6.97</td>
<td>0.5568</td>
</tr>
<tr>
<td>6</td>
<td>(75, $\infty$)</td>
<td>15</td>
<td>15.34</td>
<td>0.0075</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>100</td>
<td>100</td>
<td>4.0964</td>
</tr>
</tbody>
</table>

Now we have the $\chi^2$ value:

$$\chi^2 = (O_1 - E_1)^2/E_1 + (O_2 - E_2)^2/E_2 + \ldots + (O_6 - E_6)^2/E_6 = 4.0964$$

There are 6 categories, and hence $v = 5$. The tabulated $\chi^2$ value corresponding to $v = 5$ and $\alpha = 0.05$ is 11.07 (see Table 3). Since the calculated value does not exceed this quantity, we accept the hypothesis. Moreover, the tabulated $\chi^2$ value corresponding to $v = 5$ and $\alpha = 0.95$ is 1.1455. Since the calculated value is greater than this quantity, we have an additional justification for accepting the hypothesis (the intercreation time is exponentially distributed with mean of 40 minutes) for 5% rejection probability.

Now we can hypothesize with 95% level of significance that the process intercreation time at the start activity follows an exponential distribution with mean 40 minutes.

### 9.6 Conclusion

A variable with stochastic or not deterministic values is called a random variable. A particular value of a random variable is called a random variate. Random variables are classified according to their probability density functions.

The distribution functions introduced in this chapter can be utilized in a simulation study of a WfMS.

Uniform distribution, which is sometimes used as an approximation to a more complex distribution when a detailed simulation model is not required, is used to represent a truly random variable with the value distributed in a given range. The Normal distribution is the most common continuous distribution useful in modeling most measurement phenomena. Exponential distribution has
been used to model “sudden and catastrophic” failures and to characterize service times and interarrival times in queuing systems. **Gamma** distribution is often used to represent corresponding empirical data.

In many realistic problems the probability that an event will occur is expressed in terms of **empirical** distribution, grouped data falling into each subinterval of the sample space. The cumulative distribution of an empirical distribution can be constructed by the data.

When a random variable of interest in a simulation study does not appear to follow any of the common distributions, it will be hypothesized that the random variable follows an empirical distribution. **Whether** a random variable is represented by an empirical distribution or is known to follow a particular distribution, the analyst encounters the problem of estimating the appropriate parameters of the distribution.

If it is hypothesized that a random variable **comes** from some known common distribution, statistical methods can then be used to assess the validity of the hypothesis for a given level of significance of the test $1 - \alpha$. Student's $t$-distribution is used primarily to test differences in means of two samples selected from normally distributed populations. The $\chi^2$ distribution is used to test how well a set of observations can be represented by a given distribution function (goodness-of-fit tests). The $F$-test is used to test hypothesis that the variances of the two populations are equal. For more discussion about hypothesis test see [Maisel, 1972].
10 RANDOM VARIATE GENERATION

10.1 Introduction

*Random numbers* refer to the variates of a random variable following standard uniform distribution $U(0, 1)$. They are the basis for generating random variates of various random variables following empirical or theoretical, discrete or continuous distributions.

“The key to simulating discrete, random events is the ability to generate random numbers on a computer. A great many random numbers will be required for a typical simulation study. It is therefore essential that they be generated as quickly and efficiently as possible.” [Gottfried, 1984, p. 19]

For a WfMS simulation model, some input variables will be decided by unexpected factors and it is not realistic to set them as fixed or deterministic. Random variates obtained upon random numbers can be used for the following purposes when simulating a WfMS:

- generate activity execution/delay time;
- generate material occupation time;
- generate routing time of a work from one activity to another;
- generate intercreation time of process instances at the start activities of a process definition;
- determine role parameter if it should be decided at run-time;
- determine workflow participants among a team for execution of an activity; etc.

A random number generator on a computer is a method to generate random numbers for a simulation study. Almost every random number generator utilizes a completely determined calculation, based upon a set of unique and rigid rules, to generate a sequence of numbers. The sets of random numbers generated by a computer are called *pseudorandom numbers*.

Ideally, a pseudorandom number generator should possess all of the following desirable characteristics.

1. Randomness. First and foremost, it is essential that the generated sequence of pseudorandom numbers exhibits the same properties as truly random numbers.
2. Large period. Since all pseudorandom number generators are based upon the use of precise, deterministic formulas, every pseudorandom number sequence will eventually begin to repeat itself. The size of the nonrepeating sequence is called *period*. The period should be as large as possible. From a practical viewpoint, the period should at
leat be sufficiently large so that the random numbers do not repeat themselves during any single simulation.

3. Reproducibility. When debugging a simulation program or carrying out a parametric study (that is, varying the input data), it may be desirable to generate exactly the same sequence of random numbers during each simulation. There are other situations, however, in which different sequences of random numbers are required in a given simulation study. Therefore, the random number generator should be capable of providing both repeated and distinct random number sequences, in accordance with the wishes of the analyst.

4. Computational efficiency. Since a typical simulation study will require that a great many random numbers be generated, the random number generator should provide these numbers using as little computer time as possible. Moreover, the random number generator should not require extensive computer memory.

In actual practice, the realization of all four of these properties is quite difficult to achieve. See [Gottfried, 1984, p. 20].

10.2 Generating Random Number

10.2.1 The Power Residue Method

The power residue method (also called the multiplicative congruential method) is a simple, popular random number generator. The method makes use of the following recursive congruential relationship (see [Gottfried, 1984, pp. 25-28]):

\[ n_i \equiv a n_{i-1} \pmod{m} \]

where \( n_i \) and \( n_{i-1} \) are successive random integers, and \( a \) (the multiplier) and \( m \) (the modulus) are specified.

If we begin with some known integer constant \( n_0 \), then the repeated use of above equation results in

\[ n_1 \equiv a n_0 \pmod{m} \]
\[ n_2 \equiv a^2 n_0 \pmod{m} \]
\[ \ldots \]
\[ n_i \equiv a^i n_0 \pmod{m} \]

Thus, the successive random integers are related to the power residues of \( a \).

In order that the calculated \( n_i \) (\( i = 1, 2, \ldots \)) exhibits acceptable random behavior, it is essential that the values for \( m, n_0, \) and \( a \) are chosen in accordance
with a carefully developed set of rules. One such set of rules, which is quite commonly used, is:

1. The modulus, $m$, should be chosen as large as possible in order to maximize the period of the random number sequence. When the method is implemented on a computer having $w$ bits per word, the modulus is selected as

$$m = 2^w - 1$$

2. The multiplier, $a$, must be chosen in such a manner that the correlation between successive $n_i$'s is minimized, while at the same time obtaining the largest possible period. This can be accomplished provided the multiplier satisfies the following two conditions:

$$a \equiv 2^{\lfloor w/2 \rfloor}$$
$$a \equiv \pm 3 \pmod{8}, \text{ or } a \pmod{8} = \pm 3$$

3. The initial value $n_0$, called seed, can be any positive, odd integer whose value is less than $m$. Different seeds can be used to generate different sequences of random numbers, even though the modulus and the multiplier remain the same.

When the value for $m$, $a$, and $n_0$ are chosen in this manner, each resulting sequence of random numbers will have a period equal to $m/4$. The value of the $n_i$ ($i = 1, 2, ...$) obtained in this manner will range from 1 to $(m - 1)$. The desired uniformly distributed random variate $u_i$ ($i = 1, 2, 3, ...$) can then be obtained from

$$u_i = n_i / m$$

and so $0 < u_i < 1$.

The power residue method, with the parameters chosen in the manner indicated above, is very widely used, both for instructional purposes and for solving actual simulation problems in business and industry. There are two reasons for the method’s popularity. First, the method is able to satisfy most statistical tests for randomness; and second, it can very easily be implemented in a high-level programming language. See [Gottfried, 1984, p. 27]

But [Fishman, 1973, pp. 176-178] gave an example of bad choice of the parameters in accordance with the above rules. For more sophisticated methods to generate random numbers see [Fishman, 1973], [Gottfried, 1984], [Maisel, 1972] and [Naylor, 1966].
10.2.2 Algorithm for **Generating Random Numbers**

**Hypothesis**
A computer has a 16-bit word.

**Principle**
The power residue method:

\[ n_i \equiv a n_{i-1} \pmod{m} \]
\[ u_i = n_i / m \]

is to be implemented on the computer with \( w = 16 \) to generate random numbers. Thus it can be determined that

\[ m = 2^{w-1} = 2^{15} = 32768 \]
\[ a = 259 \equiv 2^{w/2} = 2^8 = 256, \text{ with that } a \equiv \pm 3 \pmod{8} \]

The seed \( n_0 \) is a parameter of the procedure. The new random number is generated either upon a given positive seed or the last generated number denoted by \( n \). If value of the seed is 0 or negative, it is assumed that the seed is not given; otherwise the seed is used to obtain the new number and it should be a positive, odd integer value and be less than \( m \) as well.

**Procedure \((n_0)\)**
Step 1: if \( n_0 \leq 0 \), go to Step 6;
Step 2: \( n_0 \leftarrow n_0 \pmod{m} \) (let \( n_0 \) be less than \( m \));
Step 3: if \( n_0 \pmod{2} = 1 \) (\( n_0 \) is odd), go to Step 5;
Step 4: \( n_0 \leftarrow n_0 + 1 \);
Step 5: \( n \leftarrow n_0 \);
Step 6: \( n \leftarrow a n \pmod{m} \);
Step 7: \( u \leftarrow n / m \);
Step 8: stop (return \( u \)).

**Example 10-1**
Running the procedure the first time with seed 139 and then 59 times with seed 0, the following 60 random numbers will be obtained:

\[
\begin{align*}
0.09866330078125 & \quad 0.553802490234375 & \quad 0.434844970703125 \\
0.624847412109375 & \quad 0.835479736328125 & \quad 0.389251708984375 \\
0.816192626953125 & \quad 0.393890380859375 & \quad 0.017608642578125 \\
0.560638427734375 & \quad 0.205352783203125 & \quad 0.186370849609375 \\
0.270050048828125 & \quad 0.942962646484375 & \quad 0.227325439453125 \\
0.877288818359375 & \quad 0.217803955078125 & \quad 0.411224365234375 \\
0.50711059703125 & \quad 0.341644287109375 & \quad 0.485870361328125 \\
0.840423583984375 & \quad 0.669708251953125 & \quad 0.454437255859375 \\
0.699249267578125 & \quad 0.105560302734375 & \quad 0.340118408203125
\end{align*}
\]
10.2.3 Generating Random Numbers in Basic

Basic or Visual Basic is a general-purpose programming language that is commonly available and frequently used for many business and technical applications. A random number generator is included in the language as a standard library function. Utilization of the random number generator is illustrated in the examples below.

Example 10-2
Shown below is a Basic program that first initializes the random number generator, and then generates and prints out 100 \( U(0, 1) \) random variates.

```
Randomize
For i = 1 to 100
    Let u = Rnd
    Print u
Next i
```

The statement “Randomize” initializes the random number generator. It does not need to provide a specific value for the seed. The actual random numbers are generated within the “For-To” loop by accessing the library function “Rnd”. Because no argument is given when accessing this function, the next random number in the sequence will be returned.

One run of the Basic program obtains the following 100 pseudorandom numbers:

0.8489191 0.9158093 0.9948629 0.3560297 0.1650462 
0.9979457 0.9069009 0.3945737 0.9754397 0.03868878 
0.6189001 0.005890608 0.1495456 0.7422012 0.0808726 
0.1524439 0.3624737 0.3956534 0.4938071 0.04376721 
0.5481228 0.5098485 0.3877941 0.9020896 0.6023061 
0.7098476 0.9501503 0.9779603 0.3358755 0.8755945
Different runs of the program results in different sequences of 100 pseudorandom numbers.

**Example 10-3**
This Basic program repeats sequences of 10 random numbers to a given seed \( s \) (> 0):

\[
\text{Rnd} \ (−1) \times s
\]

For \( i = 1 \) To 10
  \[ \text{Let } u = \text{Rnd} \]
  \[ \text{Print } u \]
Next \( i \)

The first “Rnd” function in the program has a negative argument \((-s)\), and so it makes the statement “Rnd” generate a fixed number to the value of seed \( s \).

Each run of the program generates the same random number sequence of 10 numbers. For example, the following results will always be obtained by running the program when \( s = 0.4253501 \):

\[
\begin{align*}
0.7326744 & \quad 0.3100755 & \quad 0.3004674 & \quad 0.09900606 & \quad 0.5817471 \\
0.9359531 & \quad 0.1535475 & \quad 0.5976273 & \quad 0.1622995 & \quad 0.2150481
\end{align*}
\]

### 10.3 Testing and Validating Random Numbers

The statistical properties of pseudorandom numbers generated by the chosen methods should coincide with the statistical properties of numbers generated by an idealized chance device that selects numbers from the unit interval \((0, 1)\)
independently and with all numbers equally likely. Clearly, the pseudorandom numbers produced by computer programs are not randomly distributed in this sense, since they are completely determined by the starting data and have limited precision. But so long as our pseudorandom numbers can pass the set of statistical tests implied by the aforementioned idealized chance device, these pseudorandom numbers can be treated as “truly” random numbers even though they are not.

Some statistical tests are introduced here to assess a pseudorandom number generator.

### 10.3.1 Frequency Test

“The frequency test is designed to test the uniformity of successive sets of numbers in the sequence. A procedure for this test is as follows.

1. Generate a sequence of $M$ (say 10) consecutive set of $N$ (say 100) random numbers each.
2. Partition the number range into intervals (say 10).
3. Tabulate the frequency within each interval for each of the $M$ groups.
4. Compare the results of the $M$ groups with each other and with the expected values (continuous uniform distribution) using the chi-square goodness-of-fit test.” [Graybeal, 1980, p. 86]

**Example 10-4**

The 100 pseudorandom numbers generated in Example 10-2 can be subdivided from the interval (0, 1) into 10 subintervals of equal width. Determining the number of the random numbers falling into each subinterval, we get the summarized table below. Expected number of observations based upon the assumed distribution $U(0, 1)$ is 100/10 for each category.
Goodness-of-Fit Test of 100 Pseudorandom Numbers

<table>
<thead>
<tr>
<th>Category</th>
<th>Subinterval</th>
<th>Number of Observations</th>
<th>Expected Number of Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(0.0, 0.1)</td>
<td>13</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>[0.1, 0.2)</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>[0.2, 0.3)</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>[0.3, 0.4)</td>
<td>13</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>[0.4, 0.5)</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>[0.5, 0.6)</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>[0.6, 0.7)</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>[0.7, 0.8)</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>[0.8, 0.9)</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>[0.9, 1.0)</td>
<td>16</td>
<td>10</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>(0.0, 1.0)</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

Calculate a $\chi^2$ statistic for this experiment and determine whether to accept or reject the hypothesis that the assumed distribution can be used to represent the data, based upon a 5% rejection probability.

The $\chi^2$ statistic is determined as

\[
\chi^2 = \frac{(O_1 - E_1)^2}{E_1} + \frac{(O_2 - E_2)^2}{E_2} + \ldots + \frac{(O_{10} - E_{10})^2}{E_{10}}
\]

\[
= \frac{(13 - 10)^2}{10} + \frac{(7 - 10)^2}{10} + \frac{(7 - 10)^2}{10} + \frac{(13 - 10)^2}{10} + \frac{(7 - 10)^2}{10} + \frac{(10 - 10)^2}{10} + \frac{(8 - 10)^2}{10} + \frac{(12 - 10)^2}{10} + \frac{(16 - 10)^2}{10}
\]

\[
= 9.8
\]

Since there are 10 categories, and hence $v = 9$. The tabulated $\chi^2$ value corresponding to $v = 9$ and $\alpha = 0.05$ is 16.92 (see Table 3). Since the calculated value does not exceed this quantity, we accept the hypothesis. Moreover, the tabulated $\chi^2$ value corresponding to $v = 9$ and $\alpha = 0.95$ is 3.325. Since the calculated value is greater than this quantity, we have an additional justification for accepting the hypothesis: the 100 variates are governed by the $U(0, 1)$ distribution.

Finally, the distinction between randomness and uniformity should be recognized. Consider, for example, the number sequence 0.05, 0.10, 0.15, 0.20, …, 0.95, 1.00. This sequence is obviously not random, though it is perfectly uniform. The test does examine randomness, in a sense, by rejecting a number sequence that is too uniform (that is, a number sequence whose calculated $\chi^2$ value is less than the tabulated value for a high rejection probability).
10 RANDOM VARIATE GENERATION 137

10.3.2 Increasing and Decreasing Runs

“The random oscillatory nature of sequences of pseudorandom numbers can be tested by ‘tests of runs.’” [Naylor, 1966, p. 60]

A run is a succession of similar events, preceded and followed by different events. In this particular test a succession of continually increasing or continually decreasing pseudorandom numbers will constitute a run. See [Gottfried, 1984, pp. 41-43].

The procedure is to count the total number of increasing and decreasing runs, and also the number of runs of length $n$, where $n = 1, 2, 3, \ldots$, etc. The observed number of runs can then be compared with the expected number of runs, where the latter values are obtained from the following expressions.

1. Total number of runs:

$$E_{TOT} = \frac{(2N - 1)}{3}$$

where $N$ is the total number of pseudorandom variates.

2. Runs of length $n$:

$$E_n = 2[(n^2 + 3n + 1)N - (n^3 + 3n^2 - n - 4)] / (n + 3)!$$

for $n = 1, 2, 3, \ldots, N - 2$, and

$$E_{N-1} = 2 / N!$$

The success or failure of the test can be determined by calculating a value for the $\chi^2$ statistic based upon the runs of length $n$.

Example 10-5

Let us apply the increasing and decreasing runs test to the 100 pseudorandom numbers presented in Example 10-2. These numbers are repeated below. Reading from left to right, we place a “+” beside each number that is greater than its predecessor, and a “−” beside each number that is less.

0.8489191 0.9158093+ 0.9948629+ 0.3560297− 0.1650462−
0.9979457+ 0.9069009− 0.3945737− 0.9754397+ 0.0386878−
0.6189001+ 0.005890608− 0.1495456+ 0.7422012+ 0.0808726−
0.1524439+ 0.3624737+ 0.3956534+ 0.4938071+ 0.04376721
0.5481228+ 0.5098485− 0.3877941− 0.9020896+ 0.6023061−
0.7098476+ 0.9501503+ 0.9779603+ 0.3358755− 0.8755945+
0.6035443− 0.8158741+ 0.6778471− 0.5325409− 0.834121+
0.7698958− 0.1285262− 0.4922713+ 0.2696272− 0.7692857+
An increasing or decreasing run can now be identified as a sequence of like signs.

There are a total of 66 runs (33 positive and 33 negative) in this example. The expected number of runs, based upon $N = 100$, is obtained from the equation as $E_{TOT} = (2N - 1)/3 = 66.3$.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$O_n$</th>
<th>$E_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>41</td>
<td>41.75</td>
</tr>
<tr>
<td>2</td>
<td>18</td>
<td>18.10</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>5.15</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1.11</td>
</tr>
<tr>
<td>5-99</td>
<td>0</td>
<td>0.23</td>
</tr>
</tbody>
</table>

The results obtained for runs of length $n$ are summarized in above table. Regrouping the data so that $E_n > 5$ for each new category, we obtain

<table>
<thead>
<tr>
<th>$n$</th>
<th>$O_n$</th>
<th>$E_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>41</td>
<td>41.75</td>
</tr>
<tr>
<td>2</td>
<td>18</td>
<td>18.10</td>
</tr>
<tr>
<td>3-99</td>
<td>7</td>
<td>6.49</td>
</tr>
</tbody>
</table>

A $\chi^2$ statistic can now be calculated as

$$\chi^2 = \frac{(41 - 41.75)^2}{41.75} + \frac{(18 - 18.10)^2}{18.10} + \frac{(7 - 6.49)^2}{6.49} = 0.0541$$

Table 3 indicates a value of 9.210 for $\nu = 2$ and $\alpha = 0.01$. Since this value exceeds the calculated value, we conclude that the given pseudorandom numbers are sequenced randomly. This conclusion is further supported by the tabulated $\chi^2$ value of 0.00201, corresponding to $\nu = 2$ and $\alpha = 0.99$. 
10.3.3 Other Tests

The reader has been familiarized with the concepts of uniformity and randomness, in the statistical sense, by describing a few of the more commonly used statistical tests. Numerous other statistical tests for randomness, uniformity, and independence have been devised. The reader should recognize the existence of these tests and should appreciate the effort that may be involved in establishing the validity of a given random number generator. See [Gottfried, 1984, p. 43].

For example, [Naylor, 1966, pp. 57-62] offers the following tests for various purposes.

- **Serial tests**: check the degree of randomness between successive numbers in a sequence.
- **The lagged product test**: measure of the independence of pseudorandom numbers is provided by a lagged product coefficient.
- **Runs up and down test and Runs Above and Below the Means test**: test the random oscillatory nature of sequences of pseudorandom numbers.
- **The Gap Test**: concerned with the randomness of the digits in a sequence of numbers.
- **The Maximum Test**: a more stringent test than the basic frequency test.
- **The Poker Test**: a special frequency test for combinations of five or more digits in a random number.

10.4 Generating Random Variates

We now turn our attention to the generation of random variates of the random variables that are governed by various distribution functions other than $U(0, 1)$ distribution. Such random variates are usually required when simulating a realistic problem situation. In fact, many realistic simulation models require the generation of several different types of random variates in order to describe the actual systems.

The following distribution functions are implemented in the Espresso simulation model.

- uniform distribution: generate time, multiple variates of a variable following empirical distribution, value of a variable defined in a routing condition, process intercreation time between two given points of time;
- normal distribution: generate time;
- exponential distribution: generate time;
- gamma distribution: generate time;
empirical distribution: generate multiple variates of a random variable following empirical distribution for choosing members to execute an activity, routing work along one of outgoing “Exclusive Choice” links of an activity, specifying role parameter, and assigning value to a variable in a routing condition;

\( U[1, n] \) distribution (an integer-valued uniform distribution in the range \([1, n]\)):

determine number of members to execute an activity;

0-1 or \( U[0, 1] \) distribution (an event happens with a given probability):

decide whether to route a work along a “Multiple Choice” link, and whether the work is divisible by multiple workflow participants of an activity.

In the following sections we will see how random numbers can be used to obtain random variates of a random variable following other distributions. The inverse transformation method will be presented, and then applied to several specific, commonly used distributions.

### 10.4.1 General Methods for Generating a Variate

Two general methods for generating a variate that is not governed by a \( U(0, 1) \) distribution are introduced here. The inverse transformation method is more popular than the rejection method because of its higher computational efficiency.

#### 10.4.1.1 The Inverse Transformation Method

Suppose that a probability density function \( f(x) \) is given, and a random variate that is governed by this probability density function is required to generate. The inverse transformation method offers a simple and straightforward approach to this problem. See [Gottfried, 1984, p. 76].

Corresponding to the given probability density function \( f(x) \), the cumulative distribution function \( F(x) \) can be obtained. Thus

\[
F(x) = \int_{-\infty}^{x} f(t)dt
\]

where \( 0 \leq F(x) \leq 1 \). Figure 10-1 shows a plot of a typical cumulative distribution function.
The cumulative distribution function is then solved for $x$. That is, if $y = F(x)$, then we can write

$$x = F^{-1}(y)$$

This expression allows us to determine the particular value of $x$ that corresponds to a given value of $y$. Let us refer to these two values as $x_0$ and $y_0$, respectively. The relationship between $x_0$ and $y_0$ is illustrated in Figure 10-1.

Now suppose that $x$ is a random variate of random variable $X$ following the given probability density function, and that $y$ is a variate of the random variable $Y$ that has corresponding value of the cumulative distribution $F(x)$. Because

$$
\text{Prob}(Y \leq y_0) = \text{Prob}(X \leq x_0) = F(x_0) = y_0
$$

Hence

$$
\text{Prob}(Y \leq y_0) = y_0
$$

which is the expression for the cumulative distribution of the standard uniform distribution. This tells us that $Y$ is uniformly distributed within the interval $[0, 1]$, regardless of the distribution of $X$. See [Gottfried, 1984, pp. 76-78].

Therefore in order to generate a value for $X$ using the inverse transformation method, we first represent $y$ by a $U(0, 1)$ random number $u$. We can then obtain the corresponding value $x$ for $X$ by evaluating the expression:

$$x = F^{-1}(u)$$

“The inverse transformation technique is useful for transforming a standard uniform deviate into any other distribution. It is particularly useful when the distribution is an empirical one.” [Graybeal, 1980, p. 89]
PART TWO: BASIC SIMULATION KNOWLEDGE

The inverse transformation method can be used only if an analytical expression for the cumulative distribution function can be obtained and solved explicitly for \( x \). There are many probability density functions for which this is not possible. An alternative technique must be used in such situations. One such method involves direct simulation of the process under consideration.

### 10.4.1.2 The Rejection Method*

Suppose that a given probability density function \( f(x) \), which governs a random variate required to generate, has a lower and upper limit to its range, \( a \) and \( b \), respectively, and an upper bound \( c \) (see Figure 10-2). The method can then be specified as follows (see [Gordon, 1978, pp. 138-140]):

1° generate two, independent \( U(0, 1) \) distributed variates \( u_1 \) and \( u_2 \);
2° compute \( x_0 = a + u_1(b - a) \);
3° compute \( y_0 = c u_2 \);
4° if \( y_0 \leq f(x_0) \), accept \( x_0 \) as the desired output; otherwise go to 1°.

The rejection method is a convenient method of generating random variates if the density function is known. However, it “has the disadvantage that two uniform variates must be calculated for each trial point, and, since, some points are rejected, more than two uniform variates are needed for the creation of each output point.” [Gordon, 1978, pp. 140]

### 10.4.2 Uniformly Distributed Random Variates

Suppose that \( x \) is a random variate of random variable \( X \) following \( U(a, b) \) distribution, where \( a < b \). The plot of the \( U(a, b) \) cumulative distribution function is presented in Figure 10-3. Let \( u \) represent a random number of a random variable following \( U(0, 1) \) distribution. From simple proportionality, we can write...
\[(x - a)/(b - a) = (u - 0)/(1 - 0)\]

or

\[x = a + (b - a)u\]

Thus it is very simple to generate a variate of a random variable following \(U(a, b)\) from a given random number provided that \(a\) and \(b\) are known.

Now suppose that \(a\) and \(b\) are integer quantities, \(a < b\), and \(X\) is a discrete, integer-valued random variable that is uniformly distributed within the interval \([a, b]\). The distribution is denoted by \(U[a, b]\). Thus \(X\) can take on the values \(a, a + 1, a + 2, \ldots, b - 1, b\). If \(u\) is a random number, then a variate \(x\) of \(X\) can be obtained by

\[x = a + \text{INT}[(b - a + 1)u]\]

In order to understand the basis for above equation, note that, since \(0 < u < 1\), then \(0 < (b - a + 1)u < (b - a + 1)\). Therefore the quantity

\[\text{INT}[(b - a + 1)u]\]

will take on the integer values \(0, 1, 2, \ldots, (b - a)\), and hence \(X\) will assume the values \(a, a + 1, a + 2, \ldots, b\), with equal likelihood.

### 10.4.3 Normally Distributed Random Variates

The normal density function cannot be integrated analytically, hence the inverse transformation method cannot be used to generate normally distributed

![Figure 10-3. \(U(0, 1)\) Cumulative Distribution](#)
random variates. The desired random variates will be generated by direct simulation.

A particularly simple technique for generating a random observation from a normal distribution is obtained by applying the central limit theorem. Since a random decimal number has a uniform distribution from 0 to 1, it has mean 1/2 and standard deviation $1/\sqrt{12}$. Therefore this theorem implies that the sum of $n$ random decimal numbers has approximately a normal distribution with mean $n/2$ and standard deviation $\sqrt{n}/12$. Thus, if $u_1, u_2, \ldots, u_n$ are a sample of random numbers, then

$$x = \frac{\sigma}{\sqrt{n/12}} \sum_{i=1}^{n} u_i + \left( \mu - \frac{n \sigma}{2 \sqrt{n/12}} \right)$$

is a random observation from an approximately normal distribution with mean $\mu$ and standard deviation $\sigma$. This approximation is an excellent one (except in the tails, or extremities, of the distribution), even with small values of $n$. Thus values of $n$ from 5 to 10 often are used; $n = 12$ also is a convenient value because it eliminates the square root terms from the above expression. See [Hillier, 1974, p. 631].

### 10.4.3.1 Algorithm for Generating a Normal Variate

**Hypothesis**

It is known that a random variable is normally distributed and with mean $\mu$ and deviation $\sigma$.

**Principle**

By applying the central limit theorem, a $N(\mu, \sigma)$ variate $x$ can be generated with the above equation, or with

$$x = \mu + \sigma Z, \text{ and }$$

$$Z = \left( \sum_{i=1}^{n} u_i - \frac{n}{2} \right) / \sqrt{n/12}$$

Here $u_1, u_2, \ldots, u_n$ are a sample of random numbers and $n$ is the size of sample. Let $n = 12$, we have

$$Z = (u_1 + u_2 + \ldots + u_{12} - 6)$$

**Procedure ($\mu$, $\sigma$)**

Step 1: generate random numbers $u_1, u_2, \ldots, u_{12}$;

Step 2: $Z \leftarrow (u_1 + u_2 + \ldots + u_{12} - 6)$
Step 3: $x \leftarrow \mu + \sigma Z$; Stop (return $x$).

### 10.4.4 Exponentially Distributed Random Variates

In order to make use of the inverse transformation method, the equation

$$F(x) = 1 - e^{-\alpha x}$$

must be solved for $x$ first. Thus

$$x = -\left(\frac{1}{\alpha}\right) \ln[1 - F(x)]$$

Since $F(x)$ is $U(0, 1)$ distributed, the quantity $1 - F(x)$ will also be $U(0, 1)$ distributed (see [Gottfried, 1984, p. 86]). Therefore

$$x = -\left(\frac{1}{\alpha}\right) \ln u$$

where $x$ is the desired exponentially distributed random variate, and $u$ is a $U(0, 1)$ random number.

#### 10.4.4.1 Algorithm for Generating an Exponential Variate

In a simulation study of a WfMS, exponential distribution can be used to generate random time with given mean time. Suppose random variable $X$ is exponentially distributed, a variate $x$ of $X$ can be generated with

$$x = - (\mu) \ln u$$

Here $\mu$ is the mean of the distribution and $u$ is the $U(0, 1)$ random number.

#### 10.4.4.2 Exponential Variates Greater than Zero*

Suppose that the variate $x$ of a random variable $X$ following exponential distribution is required to be greater than or equal to some specified positive value $x_0$ (that is, $0 < x_0 \leq x$). The equation for generating the variates must be modified to (see [Gottfried, 1984, p. 86])

$$x = x_0 - \left(\frac{1}{\alpha}\right) \ln u$$
Also, the relationship between $\alpha$ and $\mu$ now becomes

$$\alpha = 1/(\mu - x_0)$$

Notice that these relationships reduce to those presented earlier when $x_0 = 0$.

### 10.4.5 Gamma Distributed Random Variates

The probability density function for the gamma distribution cannot be integrated analytically, hence the inverse transformation method cannot be used to generate gamma random variates. We can, however, simulate the gamma process directly, by summing $\beta$ exponential random variates. Thus, we obtain

$$x = - (1 / \alpha) \ln(\prod_{i=1}^{\beta} u_i)$$

where $x$ is the desired variate of a random variable following the gamma distribution with parameters $\alpha$ and integer-valued $\beta$, and $u_i, i = 1, 2, \ldots, \beta$, is a $U(0, 1)$ random number. See [Gottfried, 1984, p. 90] and [Fishman, 1973, p. 204]

### 10.4.5.1 Algorithm for Generating a Gamma Variate

**Gamma** distribution is used to generate random times with given mean and standard deviation.

**Hypothesis**

It is known that a random variable is gamma distributed and with mean $\mu$ and deviation $\sigma$. The gamma distribution has the parameter $\beta$ as integer.

**Principle**

A gamma variate $x$ will be returned by this procedure. Because

$$\mu = \beta/\alpha \text{ and } \sigma^2 = \beta/\alpha^2 = \mu/\alpha$$

so

$$\beta = \text{INT}(\mu^2/\sigma^2) \text{ and } \alpha = \beta/\mu$$
Furthermore

\[ \ln (u_1 \cdot u_2 \cdot \ldots \cdot u_\beta) = \ln u_1 + \ln u_2 + \ldots + \ln u_\beta \]

Temporary variable \( p \) is used for keeping the value of calculation.

**Procedure \((\mu, \sigma)\)**

Step 1: let \( \beta \leftarrow \text{INT}(\mu^2/\sigma^2) \);
Step 2: If \( \beta < 1 \), let \( \beta \leftarrow 1 \);
Step 3: let \( \alpha \leftarrow \beta/\mu \);
Step 4: let \( p \leftarrow 0, i \leftarrow 1 \);
Step 5: generate random number \( u_i \);
Step 6: let \( p \leftarrow p + \ln (u_i) / \alpha \);
Step 7: let \( i \leftarrow i + 1 \); if \( i \leq \beta \), go to Step 5;
Step 8: Stop (return \( -p \)).

---

**10.4.5.2 Algorithm for Generating a Gamma Variate with Non-Integer Valued Shape Parameter**

To generate a precise gamma variate from the given mean \( \mu \) and standard deviation \( \sigma \), \( \beta \) should not always be handled as integer-valued.

**Hypothesis**

It is known that a random variable \( X \) is gamma distributed with mean \( \mu \) and deviation \( \sigma \). The gamma distribution has the parameter \( \beta \), which can be a non-integer.

**Principle**

Suppose that random variable \( X \) is gamma distributed with \( \beta \) not always being an integer. The density function is then

\[
 f(x) = \frac{\alpha^\beta x^{(\beta-1)}}{\Gamma(\beta)} e^{-\alpha x}, \quad \text{for } x \geq 0
\]

According to [Fishman, 1973, pp. 208-210], a variate \( x \) of \( X \) can be generated as described in the procedure.

Furthermore, because

\[
 \mu = \frac{\beta}{\alpha} \text{ and } \sigma^2 = \frac{\beta}{\alpha^2} = \frac{\mu}{\alpha}
\]
so

\[ \beta = \mu^2/\sigma^2 \] and \[ \alpha = \beta/\mu \]

Temporary variables \( k, Y, Z, \gamma, j, A, B \) are used in the calculation.

Procedure \((\mu, \sigma)\)

Step 1: \( \beta \leftarrow \mu^2/\sigma^2; \alpha \leftarrow \beta/\mu; \)
Step 2: \( x \leftarrow 0; Y \leftarrow 0; Z \leftarrow 0; \)
Step 3: \( k \leftarrow \text{INT}(\beta); \)
Step 4: \( \gamma \leftarrow \beta - k; \)
Step 5: if \( k = 0 \), go to Step 8;
Step 6: generate random number \( u_j, j = 1, 2, \ldots, k; X = -\ln (u_1 u_2 \ldots u_k); \)
Step 7: if \( \gamma = 0 \) (\( \beta \) is an integer), go to Step 15;
Step 8: generate random number \( u_{k+1}; Z = -\ln (u_{k+1}); \)
Step 9: \( j \leftarrow 1; \)
Step 10: generate random numbers \( u_j \) and \( u_{j+1}; \)
Step 11: \( A \leftarrow u_j^{1/\gamma}, B \leftarrow u_{j+1}^{1/(1-\gamma)}; \)
Step 12: if \( A + B \leq 1 \), go to Step 14;
Step 13: \( j \leftarrow j + 2; \) go to Step 10;
Step 14: \( Y \leftarrow A/(A + B); \)
Step 15: \( x \leftarrow 1/\alpha(x + YZ); \) stop (return \( x \)).

The procedure part from Step 9 to Step 14 is a rejection method to generate a variate following Beta distribution with parameter \( \gamma \) and \( 1 - \gamma \).

10.4.6 Empirical Distributed Random Variates

In the Espresso simulation model, empirical distributions will be used to select links and the value of a variable for routing a work, or to determine a team (for dynamic role parameter) and members of a team for executing an activity. Variates of a random variable \( X \) following an empirical distribution, as shown in Figure 9-5, can be obtained through the inverse transformation method, while the cumulative distribution as shown in Figure 9-6 can easily be constructed from \( f_j \), the given probability that the value of \( X \) will fall into the \( j \)th subinterval.

Suppose \( XL_j \) and \( XU_j, j = 1, 2, \ldots, m \), are the lower and upper interval bounds of the \( j \)th subinterval respectively. It is known that

\[ \text{Prob}(X \leq XU_j) = Y_j \]

with \( Y_m = 1 \). For simply explaining, assume \( Y_0 = 0 \).
From a \( U(0, 1) \) random number \( u \), the subinterval for the value of \( X \) can be determined by

\[
XL_j \leq X \leq XU_j \text{ if } Y_{j-1} < u \leq Y_j
\]

that is, the value of \( X \) falls into the \( j \)th subinterval. The variate \( x \) of \( X \) can then be obtained by generating a \( U(XL_j, XU_j) \) variate if \( XL_j < XU_j \); otherwise \( x = XL_j \).

### 10.4.6.1 Algorithm for Generating an Empirical Variate

**Hypothesis**

\( f_j, j = 1, 2, \ldots, m \), is the given probability that the value of random variable \( X \) will fall into the \( j \)th subinterval \( (f_1 + f_2 + \ldots + f_m = 1) \). \( XL_j \) and \( XU_j \) are the lower and upper bounds of the \( j \)th subinterval respectively.

**Principle**

The inverse transformation method is used to generate an empirical variate with the given probabilities for the subintervals.

SumProb keeps the sum of probabilities of treated subintervals.

**Procedure**

Step 1: generate random number \( u \);
Step 2: SumProb = 0;
Step 3: \( j \leftarrow 1 \);
Step 4: if \( f_j = 0 \) (\( X \) is not possible to fall into the \( j \)th subinterval), go to Step 8;
Step 5: SumProb \( \leftarrow \) SumProb + \( f_j \);
Step 6: If \( u > \text{SumProb} \), go to Step 8;
Step 7: (\( X \) falls into the \( j \)th subinterval) stop (return a \( U(XL_j, XU_j) \) variate if \( XL_j < XU_j \); otherwise return \( XL_j \));
Step 8: \( j \leftarrow j + 1 \); go to Step 4.

### 10.4.7 The \( \chi^2 \), t- and F-Distributions*

Let \( z_1, z_2, \ldots, z_n \) be variates of a variable following standard normal distribution \( N(0, 1) \). Then

\[
c = \sum_{j=1}^{n} z_j^2
\]

is a variate of a random variable following the \( \chi^2 \) distribution with \( n \) degrees of freedom. See [Fishman, 1973, p. 213].
Suppose that
\[ t = \frac{z}{\sqrt{c/n}} \]
where \( z \) and \( c \) are variates of independent random variables following \( N(0, 1) \) distribution and \( \chi^2 \) distribution (with \( n \) degrees of freedom), respectively. Then \( t \) is a variate of a random variable \( T \) following \( t \)-distribution with \( n \) degrees of freedom. See [Fishman, 1973, p. 213).

Create a shifted variable
\[ t' = \sigma t \sqrt{(n - 2)/n} + \mu \]
so that \( t' \) is a variate of the random variable \( T' \) with mean \( \mu \) and standard deviation \( \sigma \). \( T' \) has distributional appearance similar to those of \( T \) (see [Fishman, 1973, p. 213]).

Suppose that
\[ f = \frac{c_1/v_1}{c_2/v_2} \]
where \( c_1 \) and \( c_2 \) are independent \( \chi^2 \) variates with \( v_1 \) and \( v_2 \) degrees of freedom, respectively. Then the variate \( f \) is from a random variable following the \( F \)-distribution with \( v_1 \) and \( v_2 \) degrees of freedom. See [Fishman, 1973, p. 214].

### 10.5 Conclusion

A great many random numbers following \( U(0, 1) \) distribution will be required for a simulation study. The sets of random numbers generated by a computer are called pseudorandom numbers. If a general-purpose programming language, which is programmed to build up a simulation model, does not offer a pseudorandom number generator, some determined calculations, such as the power residue method, can be used to generate the random number sequence. An ideal pseudorandom number generator has the characteristics of randomness, large period, reproducibility and computational efficiency.

Whether an offered or self-programmed random generator is ideal can be assessed through different tests. The frequency test is designed to test the uniformity of successive sets of numbers in the sequence. Tests of runs can be used to test the random oscillatory nature of sequences of pseudorandom numbers. Both of these test methods are based upon the \( \chi^2 \) test.

From pseudorandom numbers, random variates of other distributions can be obtained. The inverse transformation method is used if the cumulative distribution function \( y = F(x) \), such as normal distribution and exponential
distribution functions, can be written with $x = F^{-1}(y)$. Otherwise direct simulation techniques will be used to generate variates of certain distribution, such as uniform distribution and gamma distribution.
11 SIMULATING BUSINESS PROCESSES

With the simulation model in PM, diverse process definitions can be simulated simultaneously in different Application Databases. There are a lot of input variables relevant to the process definitions. Some assumptions of cause-and-effect relationships related to these input variables are implemented in the simulation model. The simulation model can graphically display the state variables relevant to a simulated process definition on the process map and prompt process protocols as required.

11.1 Process Settings

Various input variables associated with a process definition are specified via the process settings. They are utilized in the simulation study of a WfMS to:

- let a process instance be created and run only in a certain period of time;
- create process instances according to the given distribution at each start activity of a process definition;
- route a work along a “Multiple Choice” link with a given probability;
- route a work along one of the outgoing “Exclusive Choice” links of an activity following a given empirical distribution; and
- determine the value of a variable governed by the given variate distribution.

Process Settings are used by the simulation model and saved in the Simulation Database.

11.1.1 Process Life Period Settings

A process definition is a network of activities and used for automation of certain business processes in an organization. In the real world, a business process may have a limited life period in an environment, which is represented by an Application Database in the Espresso WfMS. So it is sometimes necessary to specify for a process definition the maximum number of process instances, and/or the life beginning date as well as the life ending date during which the process instances can be created. The dialog box in Figure 11-1 is offered by PM for the specification relevant to an Application Database.

For the simulation model, the life beginning date determines when a process definition can start to be simulated. The maximum number of the process instances and life ending date determine when to stop creation of the process
instances. Creation of process instances in the Application Database stops when one of the stipulated termination conditions is met.

Life ending date works together with the simulation life beginning date. If only one process definition is simulated in a single Application Database, the default simulation beginning date is the life beginning date of the process definition in the Application Database; otherwise it is the earliest defined life beginning date of all simulated process definitions in various Application Databases.

11.1.2 Process Creation Settings

In the Espresso WfMS, a process instance in accordance with a process definition can be created at any time by the workflow participants (human or IT resources) defined to the start activities of the process definition. For the simulation study, the rule for creating process instances should be given.

Before a simulation run, the analyst should specify for each start activity of a process definition the distribution function as well as the parameters (i.e. mean time and standard deviation) for generating process intercreation time at the start activity in a certain Application Database (see Figure 11-2). According to the given function of exponential, gamma, uniform or normal distribution, the simulation model generates random variates of the process intercreation time for a start activity.

Figure 11-1. Process Life Period Settings
The distribution function can be set as fixed (deterministic) so that the process intercreation time is always the same as the given mean (such as one day), for validating the simulation model, for evaluating the simulation results, for comparing different operating policies, etc.

If a business process corresponding to a process definition is newly created, it is possible that the instances will not be generated as frequently in the initial period. So PM offers the opportunity to define specific time for the creation of initial process instances in accordance with a process definition.

When the initial process creation series are given in the settings but the numbers of process instances are not in succession in the series (for example, process 3 will be created at the beginning of a simulation run and process 6 on the tenth day as shown in Figure 11-2), the creation times for the process instances between the two neighboring given process instances but not numbered in succession (that is, process 4 and process 5), will be generated randomly according to the uniform distribution in the range of the given creation times of the two process instances (that is, according to $U(0, 10)$ distribution). Here process 4 may be determined to be created on the 3rd day and process 5 on the 7th day—the 3 and 7 are within range $[0, 10]$.

A process creation event is an arrival event that will be discussed in Section 13.4. When a process creation event occurs, a new process instance will be created in a WfMS. A process creation event is scheduled according to the process creation settings for each start activity of a process definition.
Assumption 11-1. Simulating Process Creations

For each start activity of a simulated process definition, process instances will be created in the following rules.

1. At the beginning of a simulation run: if an initial process creation series is given, process creation events from the first process instance to the last of the series are scheduled in accordance with the process definition; otherwise, just one process creation event is scheduled according to the process intercreation distribution.

2. When a process creation event occurs: if an initial process creation series is given and the created process instance number is less than that of the last of the series, no process creation event is scheduled; otherwise, a new process creation event is scheduled according to the process intercreation distribution.

3. If specific creation times for process \( i \) (\( \geq 0 \)) and process \( k \) (\( > i + 1 \)) are given as \( a \) and \( b \) respectively but that for processes \( i + 1, i + 2, \ldots, k - 1 \) are not given, the creation times for the not given are assumed following the \( U(a, b) \) distribution. Here it is assumed that \( a = 0 \) for \( i = 0 \) (when the creation time of the first process instance is not given).

From Assumption 11-1, for each start activity of a simulated process definition, a distribution function will be utilized for generating the time between two consecutive process creation events. If an initial process creation series is given, the distribution function will not be used until the last process instance with specific creation time is created.

The process creation settings are used to initiate the eventlist of the Espresso simulation model at the beginning of a simulation run. Then a simulation run can go further with a non-empty eventlist.

**Example 11-1**

Process creation settings for start activity “Register order” of the process definition in the Figure 1-3 is specified as shown in Figure 11-2.

The process creation function is an exponential distribution with mean one day. The specific creation times for processes 1, 3, and 6 are given as: process 1 and process 3 are created at the beginning time of applying the process definition; process 10 is created on the 10th day.

We simulate the process definition two times and gather the generated creation times of the first ten process instances for each run. The data are presented in the following table (1 day = 8 hours = 480 minutes):
<table>
<thead>
<tr>
<th>Process Instance No.</th>
<th>Creation Time (minutes) (Run 1)</th>
<th>Creation Time (minutes) (Run 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(given) 0</td>
<td>(given) 0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>(given) 0</td>
<td>(given) 0</td>
</tr>
<tr>
<td>4</td>
<td>3523</td>
<td>633</td>
</tr>
<tr>
<td>5</td>
<td>4579</td>
<td>2942</td>
</tr>
<tr>
<td>6</td>
<td>(given) 4800</td>
<td>(given) 4800</td>
</tr>
<tr>
<td>7</td>
<td>4939</td>
<td>6163</td>
</tr>
<tr>
<td>8</td>
<td>5220</td>
<td>6589</td>
</tr>
<tr>
<td>9</td>
<td>5877</td>
<td>7014</td>
</tr>
<tr>
<td>10</td>
<td>5896</td>
<td>7426</td>
</tr>
</tbody>
</table>

In each run, the creation time of process 2 is determined by generating a $U(0, 0)$ variate, and creation times of process 4 and process 5 are determined by generating $U(0, 4800)$ variates. From process 7 on—after the last given process instance with specific creation time (i.e. process 6), the process creation time are determined by generating a variate of the random variable following the exponential distribution with mean 480 minutes.

If the life beginning date of a process definition is given, the specific process creation time is relative to it; otherwise it is corresponding to the value of the clock, and is related to the simulation beginning date, if given.

When a simulation run begins on a date later than the defined life beginning date of a process definition, the process instances with specific creation time before the simulation run will not be simulated. For example, for the settings in Figure 11-1 and Figure 11-2, the life beginning date of the process definition is on Jan. 1, 2000 and the processes 1, 2 and 3 of the process definition will be created on this day. If a simulation run begins on Jan 2, 2000, these three process instances will not be simulated.

### 11.1.3 Routing Probability Settings

In the Espresso WfMS run-time, the person who completes an activity will decide whether to route the work associated with the activity along an outgoing “Multiple Choice” link, and/or along which of the outgoing “Exclusive Choice” links of the activity to route the work further. However the simulation model makes these routing decisions upon the probability settings to the outgoing “Multiple Choice” and “Exclusive Choice” links of an activity as shown in Figure 11-3.
According to the definitions of routing options in Chapter 3, along only one of all the outgoing “Exclusive Choice” links of an activity a work can be routed further. So the specification for the outgoing “Exclusive Choice” links of an activity is an empirical distribution for choosing one among them. That is, the sum of routing probabilities for all the outgoing “Exclusive Choice” links of an activity should be 1, or 100%.

Since a person can choose several outgoing “Multiple Choice” links after completing an activity, whether to route a work along an outgoing “Multiple Choice” link of the activity can be assumed to follow a 0-1 distribution in the simulation model. Thus routing probability of one outgoing “Multiple Choice” link of an activity dose not influence on that of other outgoing “Multiple Choice” links of the same activity.

**Assumption 11-2. Simulating a “Multiple Choice” Link**
Whether to route a work along a “Multiple Choice” link is determined by generating a variate governed by the 0-1 distribution with a given probability. The determination is independent on other outgoing “Multiple Choice” links of the same activity.
11.1.4 Variable Settings

The “Condition” link within a process definition is used for the Espresso workflow engine in the Application Database to make the routing decision automatically according to the current values of some variables, which stand for Notes fields in the document representing a process instance (see Figure 3-1). The value of a variable will be determined at run-time according to the behavior of a particular business process and can be one of the types of number, text, logic and date.

For simulation study, the variate distribution for determining values of a variable in routing conditions can be specified in the way as shown in Figure 11-4.

After the type of a variable has been decided, the probabilities for generating the values of the variable falling in different subintervals can be specified. The value is discrete if the lower and upper bound of a subinterval is the same or the upper bound is not given. The sum of probabilities for all different subintervals should be 1 or 100%. That is, a variable defined in a routing condition is stochastically and empirically distributed.

The empty lower bound of the first subinterval or the empty upper bound of the last subinterval means that there is no lower or upper bound to the subinterval. For example, in Figure 11-4, the first subinterval means \((-\infty, 1000)\) and the last subinterval \((100000, +\infty)\), since the type of a variable is numeric.

Figure 11-4. Process Variable Settings
If the variate distribution of a variable is specified as activity-dependent, the variate of the variable defined in an outgoing “Condition” link of an activity will be generated again as the activity is completed; otherwise the variate of a variable is generated only once for a process instance, no matter at which activity.

**Assumption 11-3. Simulating an Activity-dependent Variable**

If a variable is activity-dependent, the value of the variable will be generated again when a work at an activity is completed and the variable is contained in the formula of a “Condition” outgoing link of the activity; otherwise the value of the variable will be generated only if it has never been generated for the associated process instance.

It is possible to allow multiple values separated with “;” to be assigned to a Notes field. In order to simulate this case, the system analyst should specify in the dialogbox as shown in Figure 11-4 that the variable is allowed with multi-values.

**Assumption 11-4. Simulating a Variable with Multiple Values**

If a variable is allowed to be assigned with multiple values in a WfMS, an assignment of the multiple values is determined in the following steps:

1° the number of values \( n \) is generated by \( U[1, m] \), here \( m \) is the number of subintervals of the empirical distribution followed by the variable;

2° let \( n \) be the same as \( M \), if generated \( n \) is larger than \( M \), the number of subintervals with non-zero probability;

3° according to the distribution, the assignment of \( n \) values will be generated so that they fall in \( n \) different subintervals respectively.

For example, from the settings in Figure 11-4, assignment of multiple values is allowed for variable “LoanAmount”. Five different subintervals are specified with relevant probabilities. One variate generated according to the distribution can be “82900:1693:59277886” —three values are included in the assignment: one falls in the fourth subinterval (10000, 100000), one in the second subinterval (1000,10000), and another in the fifth subinterval (100000, +∞).

Usually, the subintervals with zero probability, such as the first subinterval of the setting in Figure 11-4, should not be specified in the distribution settings. But this increases the number of subintervals of the empirical distribution of a variable and thus increases the number of multiple values in a value determination of the variable according to Assumption 11-4.

Before a simulation run, types and variate distributions of all variables defined in different routing condition formulas should be specified as they are in the real world. The simulation model can determine whether to route the work along the outgoing “Condition” links of an activity, if values of the variables in all the condition formulas can be or have been generated.
11.1.5 Process Stop Settings

If a routing decision activity has only outgoing links of “Multiple Choice” and “Condition” routing options, it can happen during simulation that neither a link will be chosen nor a routing condition is met for a work at the activity to flow further. The system analyst can decide via the dialog box shown in Figure 11-3 whether a work can stop at the activity.

Suppose a work cannot stop at a routing decision activity. When the activity is completed, the simulation model will try repeatedly to choose (during simulation) or let the system analyst to choose (during animation) “Multiple Choice” links (if any), or to determine values of variables defined in the routing formulas of all outgoing “Condition” links of the activity (if the variables are activity-dependent), till along at least one outgoing link of the activity the work flows further. In this case, the statistical simulation result of the choice percentage will be larger than the specified probability for each outgoing “Multiple Choice” link.

Assumption 11-5. Simulating Stop of a Process Work

At an activity combined only by “Multiple Choice” and “Condition” outgoing links, it is possible that the work associated with a process instance stop there. If this is not allowed, the selection for the “Multiple Choice” links and/or determination of values of activity-dependent variables in the formulas of the “Condition” links will be repeated until the work can be routed further.

The simulation model can prompt with a message about a stopped work and meanwhile let the relevant activity on the process map be marked.

If a work is not allowed to stop at an activity which has only outgoing links of “Condition” links and no variable in the routing formulas of these links is activity-dependent, it can happen that the work can never run further. So some work items completed there are locked and the associated process instance can not be terminated. The system analyst must keep in mind that to change values of mere activity-dependent variables in a routing formula may not let a formula evaluate to TRUE, although a lot of computer time is taken for repeating the determination. To let a simulation run further, the simulation model repeats a maximum of 99 times for the determination of routing of a work from such an activity.

11.2 Graphic Process States

During simulation, some values of state and statistic variables relevant to a process definition are displayed beside each activity icon as shown in Figure
11-5. A work flowing from one activity to another can be animated by a document icon with the number of the associated process instance within it. For each activity, the displayed state variables are

- the number of the last coming or routed process instance, and
- total number of running (i.e. existing) activity instances in the WfMS;

The displayed statistic variables are

- total number of created activity instances, and
- total number of completed activity instances.

For the example in Figure 11-5, a work associated with process 2 in accordance with the process definition is just flowing along the link from activity “Complete order” to activity “Notification”. For activity “Complete Order”, the last treated process instance is the second process instance; there is one instance of the activity running in the simulated system; total number of created instances of the activity is three and two of them have been completed. It is known that six process instances in accordance with the process definition have been created according to the data beside the icon of start activity “Register order”; and that one process instance has been terminated from the data beside the icon of end activity “Notification”. As a whole it is known that there are five running process instances in accordance with the process definition: three process instances are at activity “Check order”, one is at
activity “Complete order”, and one is just being routed from activity “Complete order” to activity “Notification”.

That is, routing document icons animate how work items associated with different process instances flow from activity to activity on the process map. The data beside the activity icons exhibit the dynamic states of the simulated process definition.

11.3 Data Structure of Simulated Work Items

If parallel activities are defined in a process definition, it is possible that during simulation, parallel work items associated with the same process instance exist in the simulated system. Parallel work items are simultaneously simulated and their behavior can be graphically displayed.

For simulating split-join enactment and for reporting a process protocol (see Section 13.3.1), a protocol of each simulated work should be maintained. A work protocol keeps the start activity, where the associated process instance is created, all in parallel or in sequential completed activities, and the current activity of the work. A work protocol will be copied when the work is split into multiple parallel work items; two work protocols will be merged when join happens.

11.3.1 Work Record

Record is “a group of related data, facts, or fields of information that is treated as a unit”. [Weinberg, 1980, p. 317]

Data of a work associated with a simulated process instance is kept in a record, called work record, which contains simulation information about the process instance and the activity instance. Parallel work items associated with the same process instance, such as the work items in records 2, 4, and 5 shown in Figure 11-6, are connected with one another to a loop (a closed chain) by record parallel pointers. A parallel work loop helps to locate all parallel work

Figure 11-6. Parallel Work Record Loop
Example 11-2
Suppose five work records 1, 2, 3, 4, and 5, are used to keep three simulated process instances as shown in Figure 11-6. The work items in records 2, 4, 5 are associated with one process instance and are connected by record pointers into a loop 2→4→5→(2). That is, the work in record 1 is associated with a process instance, the work in record 3 is associated with another process instance, and the parallel work items in records 2, 4, and 5 are associated with a common process instance.

Now suppose that the work in record 4 will be split into two work items, and the new one is kept in record 6. That is, the new work in record 6 is parallel with that in record 4. The pointer from record 4 connecting to record 5 is removed and one from record 4 connecting to record 6 and another from record 6 connecting to record 5 is added, as shown in Figure 11-7. At the present, the four parallel work items in loop 2→4→6→5→(2) are associated with the same process instance.

Now suppose from the work records in Figure 11-7, the parallel work in record 2 will be removed. Before the work is removed, the pointer connecting record 5 to record 2 will be altered to connect to record 4, which is originally connected from record 2, and the pointer from record 2 connecting to record 4 will be deleted. The existing work records will be connected as that shown in Figure 11-8. At this moment, there are five work items in records 1, 3, 4, 5, and 6 respectively, associated with three simulated process instances.
The data structure for keeping parallel work items has been used by the algorithms for Detecting Deadlocks and Transmitting Work to Activity in Chapter 6 for finding out parallel work items associated with the same process instance.

11.3.2 Work Protocol

A work protocol is a protocol of a work associated with a simulated process instance. It consists of protocol records of completed activities and the current activity in the data structure of parallel chain as shown in Figure 11-9. The work record has a pointer connecting to the protocol record of the activity at which the work is currently located.

Each protocol record has workflow control data as well as application data about an activity. The most important data of a protocol record used by the simulation model are the identical number of the activity (displayed in a protocol record in Figure 11-9), work arrival time at the activity, and the predecessor set of the activity. The predecessor set of a protocol record consists of pointers connecting to the protocol records of the predecessor activities that have been completed. The arrival time contained in a protocol record is used by the simulation model to spot the split activities according to the protocols of two parallel work items and to determine critical activities on a work protocol.

Example 11-3

For the process definition in Figure 5-5, suppose a work is now at activity 4. The protocol can be represented by the data structure as shown in Figure 11-9. Protocol records 1, 2, 3 and 5 have data of activities 1, 2, 3, and 4 respectively. Record 5, the last record of the protocol, has a predecessor set of {2, 3}. That is, the work at activity 4 was routed from activities in protocol records 2 and 3 (i.e. activity 2 and activity 3). Both records 2 and 3 have a predecessor set of {1} and record 1 keeps the data of activity 1. So activity 2 and activity 3 was executed after activity 1 was completed. Record 1 has an empty predecessor set and thus activity 1 is the start activity of the work kept in the work record.
Based upon the work protocol, a report about the execution thread (sequential as well as parallel) of a process instance, called process protocol, can be generated from the start activity to a specified activity (see Chapter 13).

11.3.3 Algorithms Relevant to Simulated Work Items

The algorithms dealing with parallel work items as well as work protocols are explained here. Figure 11-10 illustrates the relationships between the algorithms and work records as well as protocol records.

Figure 11-10. Relationships between Algorithms of Simulated Work Items

11.3.3.1 Algorithm for Connecting a Parallel Work

This procedure is used when splitting a work into several parallel work items.

Hypothesis
Here records $k$ and $i$ are parameters of the procedure. The work in record $k$ is newly created and it is parallel with that in record $i$. Suppose ParallelPointer($j$) represents the record pointer of work record $j$ for connecting parallel work items into a loop.
Principle
Because the work in record $k$ is parallel with that in record $i$, record $k$ will be connected to the parallel work record loop with record $i$. (See Figure 11-6 and Figure 11-7, where the work in record 6 is newly created and is parallel with that in record 4 and so is added to the parallel loop.)

Procedure $(k, i)$
Step 1: if $\text{ParallelPointer}(i) = 0$ (the work in record $i$ is originally not parallel), go to Step 3;
Step 2: $\text{ParallelPointer}(k) \leftarrow \text{ParallelPointer}(i)$; go to Step 4;
Step 3: $\text{ParallelPointer}(k) \leftarrow i$;
Step 4: $\text{ParallelPointer}(i) \leftarrow k$; stop.

11.3.3.2 Algorithm for Disconnecting a Parallel Work

This procedure is called by the algorithm for Eliminating a Work Record (see Section 11.3.3.7) before deleting a work associated with a simulated process instance.

Hypothesis
Suppose $\text{ParallelPointer}(j)$ represents the record pointer of work record $j$ for connecting a parallel work loop.

Principle
Here record $i$ is a parameter of the procedure. This procedure removes the work kept in work record $i$. (See Figure 11-7 and Figure 11-8, where work in record 2 is removed.)
Temporary variable $\text{RecordNo}$ is used for keeping the current treated record.

Procedure $(i)$
Step 1: if $\text{ParallelPointer}(i) = 0$ (the work in record $i$ is not parallel), stop;
Step 2: $\text{RecordNo} \leftarrow \text{ParallelPointer}(i)$;
Step 3: if $\text{ParallelPointer}(\text{RecordNo}) = i$, go to Step 5;
Step 4: $\text{RecordNo} \leftarrow \text{ParallelPointer}(\text{RecordNo})$; go to Step 3;
Step 5: if $\text{RecordNo} = \text{ParallelPointer}(i)$ (there is only one remained parallel work to the work in record $i$),
  let $\text{ParallelPointer}(\text{RecordNo}) \leftarrow 0$ and stop;
Step 6: $\text{ParallelPointer}(\text{RecordNo}) \leftarrow \text{ParallelPointer}(i)$; stop.
11.3.3.3 Algorithm for **Copying a Work Protocol**

This procedure will be called when splitting a work into two or more parallel work items. For each new created work, a work protocol from the start activity to the current activity should be generated according to the protocol of the original work.

**Hypothesis**
Suppose Predecessors($j$) represents the predecessor set of protocol record $j$.

**Principle**
Here protocol records $i$ and $k$ are parameters of the procedure. This self-called procedure is called by another procedure when creating a parallel work at the activity registered in protocol record $i$. The work protocol to the last protocol record $i$ (protocol record $i$ as well as that of all previous completed activities) should be copied. The last protocol record of the new work protocol will be added to the predecessor set of protocol record $k$.

Global set HasCopied keeps copied records and must be clear to empty $\phi$ before the procedure is called by another procedure.

Temporary set RestRecords is used to keep not treated protocol records belonging to the predecessor set of protocol record $i$. Temporary variable $j$ refers to the current new created protocol record.

**Procedure ($i, k$)**
Step 1: if $i \notin$ HasCopied, go to Step 3;
Step 2: (record $i$ has been copied, say to protocol record $j$) go to Step 10;
Step 3: get a new protocol record, say protocol record $j$;
Step 4: copy the content of record $i$ to protocol record $j$; Predecessors($j$) $\leftarrow \phi$;
Step 5: RestRecords $\leftarrow$ Predecessors($i$);
Step 6: if RestRecords = $\phi$, go to Step 9;
Step 7: remove an element, say protocol record $p$, from RestRecords;
Step 8: call the procedure self with parameters $p$ and $j$; go to Step 6;
Step 9: HasCopied $\leftarrow$ HasCopied $\cup \{i\}$ (protocol $i$ has been copied, say to protocol record $j$);
Step 10: Predecessors($k$) $\leftarrow$ Predecessors($k$) $\cup \{j\}$; stop.

11.3.3.4 Algorithm for **Eliminating Protocol Records**

This procedure is called by the algorithm for Eliminating a Work Record (see Section 11.3.3.7).
Hypothesis
Here protocol record $i$ is a parameter of the procedure. Suppose Predecessors($j$) represents the predecessor set of protocol record $j$.

Principle
This procedure is a self-called procedure. The protocol part to protocol record $i$, i.e. protocol record $i$ as well as all protocol records of previous activities, will be released.

Global set HasReleased, which keeps just released protocol records, must be clear to empty before the procedure is called by another procedure.

Temporary set RestRecords is used for keeping not treated protocol records belonging to the predecessor set of protocol record $i$.

Procedure ($i$)
Step 1: if $i$ ∈ HasReleased, stop;
Step 2: RestRecords ← Predecessors($i$);
Step 3: if RestRecords = $\emptyset$, go to Step 8;
Step 4: remove an element, say protocol record $p$, from RestRecords;
Step 5: if $p$ ∈ HasReleased, go to Step 3;
Step 6: call the procedure self with parameter $p$;
Step 7: go to Step 3;
Step 8: release protocol record $i$;
Step 9: HasReleased ← HasReleased $\cup \{ i \}$; stop.

11.3.3.5 Algorithm for Checking an Activity Split in a Protocol

This procedure is called by the algorithm for Getting All Split Activities in Protocols (see Section 11.3.3.6) for merging protocols of two parallel work items associated with the same process instance.

Hypothesis
It is known that the activity in protocol record $i$ is parallel with an activity in the part work protocol to protocol record $k$.

Suppose Predecessors($j$) represents the predecessor set of protocol record $j$.

Principle
Protocol records $i$ and $k$ are parameters of the procedure. This self-called procedure checks according to arrival time whether the activity in protocol record $i$ is a split activity in the part work protocol to protocol record $k$. If so, the protocol record of the split activity in the protocol will be returned; otherwise 0 will be returned.
Procedure \((i, k)\)
Step 1: if the activity in protocol record \(i\) is not the same as that in protocol record \(k\), go to Step 4;
Step 2: if the work arrival times at the both activities are not the same (the activity kept in protocol record \(k\) is not the split activity), go to Step 4;
Step 3: stop (return \(k\));
Step 4: \(\text{RestRecords} \leftarrow \text{Predecessors}(k)\);
Step 5: if \(\text{RestRecords} = \emptyset\), stop (return 0);
Step 6: remove an element, say protocol record \(p\), from set \(\text{RestRecords}\);
Step 7: \(r \leftarrow \text{the procedure self with parameters } i \text{ and } p\); if \(r \neq 0\), stop (return \(r\));
Step 8: go to Step 5.

11.3.3.6 Algorithm for Getting All Split Activities in Protocols

This procedure is called by the algorithm for Eliminating a Work Record (see Section 11.3.3.7).

Hypothesis
It is known that the work with part of the protocol to protocol record \(i\), the first parameter of the procedure, is parallel to the work with the part protocol to protocol record \(k\), the second parameter of the procedure.

Suppose \(\text{Predecessors}(j)\) represents the predecessor set of protocol record \(j\).

Principle
Protocol records \(i\) and \(k\), as well as set \(U\) are parameters of the procedure. This self-called procedure puts a set of data about split activities of the two parallel work, say \(i\#p\#r\), to set \(U\), if the activity in protocol record \(p\), \(p \in \text{Predecessors}(i)\), is also in protocol record \(r\) that belongs to the work protocol to protocol record \(k\). That is, the activity in protocol records \(p\) and \(r\) is a split activity of the two parallel work items.

Before the procedure is called by another procedure, set \(U\) is assigned with empty \(\emptyset\).

Procedure \((i, k, U)\)
Step 1: \(\text{RestRecords} \leftarrow \text{Predecessors}(i)\);
Step 2: if \(\text{RestRecords} = \emptyset\), stop (set \(U\) has the returned value);
Step 3: remove an element, say protocol record \(p\), from set \(\text{RestRecords}\);
Step 4: \(r \leftarrow \text{Checking an Activity Split in a Protocol } (p, k)\);
Step 5: if \(r \neq 0\) (at the activity in protocol record \(r\), the work has been split), let \(U \leftarrow U \cup \{i\#p\#r\}\);
Otherwise (activity in protocol record \( p \) is not a split activity of the two parallel work),
call the procedure self with parameters \( p, k, \) and \( U \);

Step 6: go to Step 2.

### 11.3.3.7 Algorithm for Eliminating a Work Record

A work record will be eliminated when the work is either terminated/stopped at an end activity of a process definition, or joined at a join activity.

**Hypothesis**
Suppose \( \text{Predecessors}(j) \) represents the predecessor set of protocol record \( j \) and \( \text{ProtocolPointer}(d) \) stands for the pointer to the last protocol record which keeps the data of current activity of the work in record \( d \).

**Principle**
Here work records \( i \) and \( k \) are parameters of the procedure and work record \( i \) will be released. If \( k \) is not null, it’s known that the work items respectively in records \( i \) and \( k \) are parallel (associated with the same process instance) and the work in record \( i \) is just coming to the join activity where the work in record \( k \) is waiting for joining (both work items are at the same activity now). So the protocol of the work in record \( i \) should be joined to that in record \( k \).

For example, for the process definition in Figure 11-11, the work in record \( k \) came from activity 5 and is waiting at activity 6. Now the parallel work in record \( i \) comes from activity 4 to activity 6. Protocols of the two work items are shown in Figure 11-12. When the two work items are joined, the protocols of the two work items must be joined into one. After merging the protocol of the work in work record \( i \) to that in work record \( k \), the protocol of the work in work record \( k \) becomes as that shown in Figure 11-13. Work record \( i \) as well as three protocol records that are not connected in the new work protocol will be released.

![Figure 11-11. Process Definition—Eliminating a Work Record](image-url)
Procedure \((i, k)\)

Step 1: \( r \leftarrow \text{ProtocolPointer}(i) \) (\( r \) is the last protocol record of the work in record \( i \));

Step 2: if \( k \neq 0 \), go to Step 4 (to join the protocol of the work in record \( i \));

Step 3: (the work in record \( i \) will not be joined) \( \text{HasReleased} \leftarrow \emptyset \); call Eliminating Protocol Records \((r)\); go to Step 16;

Step 4: \( q \leftarrow \text{ProtocolPointer}(k) \) (\( q \) is the last protocol record of the work in record \( k \));

Step 5: if Predecessors\((q) = \emptyset \) (activity in protocol record \( q \) has no predecessor and so no protocol to join), go to Step 16;

Step 6: \( \text{SplitActivityRecs} \leftarrow \emptyset \); call Getting All Split Activities in Protocols with parameters \( r, q, \text{SplitActivityRecs} \) (set SplitActivityRecs contains message of all split activities where the work has been split);

Step 7: \( \text{HasReleased} \leftarrow \emptyset \) (clear the set that will be used in Eliminating Protocol Records latter);

Step 8: if SplitActivityRecs = \( \emptyset \), go to Step 13;

Step 9: remove an element, say \( n\#p\#s \), from set SplitActivityRecs (protocol record \( n \) keeps the successor of the activity in protocol record \( p \). The activities in protocol records \( p \) and records \( s \) are the same split.

**Figure 11-12.** Eliminate a Parallel Work Record—before

**Figure 11-13.** Eliminate a Parallel Work Record—after
activity. Protocol records \( n \) and \( p \) are in the protocol to record \( r \), and record \( s \) is in the protocol to record \( q \);

Step 10: \( \text{Predecessors}(n) \leftarrow \text{Predecessors}(n) - \{p\} \cup \{s\} \) (switch a previous record of protocol record \( n \) from record \( p \) to record \( s \));

Step 11: call \text{Eliminating Protocol Records} \( (p) \) (release the protocol part to protocol records \( p \));

Step 12: go to Step 8;

Step 13: \( \text{Predecessors}(q) \leftarrow \text{Predecessors}(q) \cup \text{Predecessors}(r) \) (the protocol of the work in record \( i \) is added to the predecessor set of the work in record \( k \));

Step 14: \( \text{Predecessors}(r) \leftarrow \phi \);

Step 15: \( \text{HasReleased} \leftarrow \phi \); call \text{Eliminating Protocol Records} \( (r) \) (return the protocol record pointed by work record \( i \));

Step 16: call \text{Disconnecting a Parallel Work} \( (i) \);

Step 17: release work record \( i \).

### 11.3.3.8 Algorithm for Getting Process Critical Paths

This procedure is used when displaying a process protocol, an execution thread of a process instance (see Section 13.3.1). The critical paths of a work from the start activity where the process instance was created to the current activity will be determined.

#### Hypothesis

Here work record \( i \) is a parameter of the procedure. Suppose that \( \text{ProtocolPointer}(i) \) represents the pointer connecting to the last protocol record which keeps the data of current activity of the work in work record \( i \), \( \text{Predecessors}(j) \) (with protocol record \( j \) belongs to the protocol to protocol record \( \text{ProtocolPointer}(i) \)) represents the predecessor set of the activity in protocol record \( j \), and \( \text{ArrivalTime}(p), p\in \text{Predecessors}(j) \), stands for the time the work arrived to the activity kept in protocol record \( p \).

\( \text{OnCritical}(j) \) keeps the result of whether the activity in protocol record \( j \) is on critical path of the work.

#### Principle

In this procedure \( \text{OnCritical}(j) \) will be set to TRUE if the activity in protocol record \( j \) is on the critical path of the work in record \( i \).

A self-called sub procedure with the parameter of protocol record \( j \) will be called by the main procedure. It is known that the activity in protocol record \( j \) is on a critical path of the work in record \( i \). The sub procedure is to determine whether predecessors of the current activity are on a critical path.

In the sub procedure, temporary set \( \text{RestRecords} \) is used to keep not treated protocol records belonging to the predecessor set of protocol record \( j \). Set
LastArrivals keeps predecessor protocol records with the last arrival time kept in variable LastArrivalTime. The activities in set LastArrivals can be on a critical path.

**Main Procedure (i)**
Step 1: $j \leftarrow \text{ProtocolPointer}(i)$;
Step 2: OnCritical($p$) $\leftarrow$ FALSE, with that protocol record $p$ belongs to the protocol to protocol record $j$;
Step 3: call the sub procedure with parameter $j$;
Step 4: stop (OnCritical() has been determined).

**Sub Procedure (j)**
Step 1: if OnCritical($j$) = TRUE (the activity in protocol record $j$ has been treated), stop;
Step 2: OnCritical($j$) $\leftarrow$ TRUE;
Step 3: RestRecords $\leftarrow$ Predecessors($j$);
Step 4: LastArrivalTime $\leftarrow$ 0; LastArrivals $\leftarrow$ $\phi$;
Step 5: if RestRecords = $\phi$, go to Step 10;
Step 6: remove an element, say protocol record $p$, from RestRecords;
Step 7: if ArrivalTime($p$) = LastArrivalTime (the activity in protocol record $p$ is one of the last arrived), let
   LastArrivals $\leftarrow$ LastArrivals $\cup \{p\}$;
Step 8: if ArrivalTime($p$) $>$ LastArrivalTime (the activity in protocol record $p$ is the last arrived), let
   LastArrivalTime $\leftarrow$ ArrivalTime($p$) and
   LastArrivals $\leftarrow \{p\}$;
Step 9: go to Step 5;
Step 10: (all the activities in protocol records in set LastArrivals are on the critical paths) RestRecords $\leftarrow$ LastArrivals;
Step 11: if RestRecords = $\phi$, stop;
Step 12: remove an element, say protocol record $p$, from RestRecords;
Step 13: call the sub procedure self with parameter $p$;
Step 14: go to Step 11.

### 11.4 Conclusion

For simulating how a process definition will behave in the run-time WfMS, the following input variables can be specified to a process definition for the simulation model:

- the time period when the process instances can be created in accordance with the process definition, or the maximum number of the process instances to be created;
• the process intercreation distribution as well as an initial process creation series for each start activity (based upon the specification, the eventlist of the simulation model will be initiated at the beginning of a simulation run);
• the variate distribution of a variable defined in a routing condition;
• the empirical distribution for routing a work along one of all outgoing “Exclusive Choice” links of an activity;
• the probability for routing a work along a “Multiple Choice” link;
• whether a work can stop at an activity with only “Multiple Choice” and
• whether the values of a condition variable are activity-dependent and multi-values allowed.

During a simulation run, graphical information about routing work items are animated on a process map. Beside the icon of each activity, the number of the last dealt process instance, total number of running activity instances, total number of created activity instances, and total number of completed activity instances are presented.

Parallel work items associated with a process instance can be simultaneously simulated. For handling the split-join enactment and generating process protocol reports, the data structure of the parallel running work items was discussed in this chapter.
12 SIMULATING RESOURCES

To run a business process, a lot of resources as well as costs are required. In the Espresso WfMS, the resources for executing an activity within a process definition are specified with the definition of the activity. The workflow participants of an activity are determined from the editor assigned to the activity.

For the simulation model, the input variables related to an organization are specified to an Organization Database or a Notes Organization Directory, where human and material resources are defined. Some resource-relevant assumptions are implemented in the simulation model.

During simulation, states of the simulated resources can be graphically displayed in a resource window. The waiting queue of a simulated resource will be established for gathering relevant data and representing the worklist of the resource. The dummy participant of an editor acts as one of the workflow participants of the activity if the editor assigned to the activity can not be perfectly resolved to the simulated workflow participants.

12.1 Resource Specification

In the Espresso WfMS, three kinds of resources can be determined or directly specified for executing an activity within a process definition and can be simulated in PM.

- **Human resource**: a person defined in an Organization Database or a Notes Organization Directory. A person can be a member of a team defined in the databases.
- **Notes agent** (IT resource): a procedure of actions such as filling documents, sending mail, looking for particular topics, archiving older documents, manipulating field values, bringing data in from other applications, etc., on the pre-selected set of documents in an Application Database. “Agents enable you to automate frequently performed processes, eliminating tedious administration tasks and speeding your business application. Agents can be triggered by time or events in a business application.” [Toulemonde, 1998, p. 22]
- **Material resource**: a non-shared machine, device, or tool defined in an Organization Database. It can be used for the execution of the activity.

Human resources and scheduled or mail-triggered Notes agents can be workflow participants that are assigned directly or through a team to an activity within a process definition. Material resources are used for the execution of some activities.
Upon the specification of the resource requirement for the execution of an activity, the system performance criteria relevant to various resources, such as costs, resource utilization, etc., are simulated and calculated.

12.1.1 Workflow Participant Specification

In the Espresso WfMS, workflow participants who can undertake an activity is assigned by one of the eight kinds of *editors* via the PM dialog box as shown in Figure 12-1.

- **Anyone**: any person defined in an Organization Database and/or a Notes Organization Directory.
- **Computed**: the organizational roles will be read from the given Notes field, the value of which can be computed at run-time.
- **Given people**: a number of people defined in an Organization Database and/or a Notes Organization Directory. They are fixed workflow participants for the activity.
- **Group**: an organizational role defined in a Notes Organization Directory.
- **Workgroup**: an organizational role defined in an Organization Database.
- **Department**: an organizational role defined in an Organization Database, with the specification of either including or excluding sub-departments.
• **Role**: an organizational role defined in an Organization Database, with the specification of either a fixed parameter or a dynamic parameter reading from the given Notes field at run-time.

• **Notes agent**: an IT resource defined in an Application Database, which is able to automatically execute the work associated with the activity.

At build-time, an activity is assigned to an editor; at run-time, a work associated with the activity will be undertaken by the workflow participants resolved from the editor. Figure 12-2 presents the relationships between editor, organizational roles and workflow participants in the Espresso WfMS—in which databases the organization roles are defined and how an editor will be resolved to workflow participants of people or a Notes agent. The field specified for editor “Computed” can be filled with a combination of several organizational roles at run-time. Any person belonging to an organizational role can be a workflow participant to execute an activity. Like a person, a Notes agent can also be directly assigned to an activity as a workflow participant.

If the workflow participant is an IT resource (such as Notes agent in the Espresso WfMS), the capacity of CPU, where IT resources run, may influence activity duration. But this is not considered in the Espresso simulation model.

**Assumption 12-1. Simulating a Notes Agent**
The capacity limit of a Notes agent is not simulated.
For each activity, the stand-in of the editor can be specified. An activity can have no stand-in, or have what defined in the Organization Database, or be given for the activity. The given stand-in for an activity is specified in the same way as that of an editor.

The stand-in belongs to exception handling and so will not be simulated, just as ad hoc workflow within the Espresso WfMS is not simulated.

<table>
<thead>
<tr>
<th>Assumption 12-2. No Simulation of a Stand-in</th>
</tr>
</thead>
</table>
The stand-in of an activity is not considered in the simulation model.

12.1.1.1 Simulating Editor “Computed”

In the simulation model it is disregarded that a Notes field specified for editor “Computed” can be filled with a team organizational role such as a workgroup in the Espresso WfMS, because any organizational role will be eventually resolved to individual workflow participants of people.

<table>
<thead>
<tr>
<th>Assumption 12-3. Simulating Editor “Computed”</th>
</tr>
</thead>
</table>
a number of people are directly resolved from editor “Computed” as the workflow participants of an activity.

Initiator of a process instance is the workflow participant who creates at runtime the process instance at a start activity of a process definition. The initiator of a process instance will be kept in the field “wfInitiator” of the document representing the process instance in the Espresso Application Database.

When scheduling a process creation event at a start activity of a process definition, one of the workflow participants resolved from the editor assigned to the start activity is determined as the initiator of the process instance.

If the start activity can be executed only by the initiator, the activity should be assigned by editor “Computed” with field “wfInitiator”, instead of by editor

In the Espresso WfMS, the workflow participants of the predecessors of an activity instance are kept in Notes field “wfPrevMember” of the document representing the activity instance. Field “wfPrevMember” can be specified for editor “Computed” to an activity. The workflow participants of the activity instance are then all those workflow participants who have executed the predecessor activities.
Assumption 12-4. Simulating Workflow Participants of the Predecessors
The workflow participants for an activity assigned to editor “Computed” with field “wfPrevMember” are resolved as the whole of the workflow participants of all the predecessor activities. If the activity has no predecessor, the workflow participant of the activity is the initiator of the process instance.

12.1.1.2 Team Editor

Editors “Given people”, “Group”, “Workgroup”, “Department” and “Role” can be resolved to a team of people. Workflow participants specified via editor “Computed” can also be filled with one or more of the teams at run-time. So these editors are called team editors.

12.1.1.2.1 Activity Completion Specification

In the Espresso WfMS, if an activity is assigned by a team editor, the workflow engine will allocate a work associated with the activity to the worklists of all members of the team, and so any team member can execute the activity. Thus it is necessary to specify through one of the three options which or how many members resolved from a team editor should complete an activity.

- **Exact**: specified number of members of the team should complete the activity.
- **All**: all members of the team should complete the activity.
- **Given**: specified members of the team should complete the activity.

To complete an activity within a process instance in run-time of the Espresso WfMS is for a workflow participant to issue in the Application Database a command (via a button or a menu) of completing the activity, and routing the work of the associated process instance further to some successor activities or terminating/stopping the process instance if the activity is an end activity.

In the real world system, if an activity is assigned to a team editor, a team member who executes the activity and a member who completes the activity may undertake the activity in different ways. But in the simulation model, the determined workflow participants, including all specified for completing the activity, are simulated to execute the activity in the same way for such as material occupation, handling delay, execution/delay time determination, etc. That is, it is ignored in the simulation model, whether a workflow participant is
just to complete an activity, only to execute the activity, or both to execute and
to complete the activity.

**Assumption 12-5. Simulating Completion of an Activity**
A person who completes an activity is simulated in the same way as a
workflow participant who executes the activity.

If **exact** number of members of a team editor is specified for the activity
completion, the members who should complete the activity are uncertain.

### 12.1.1.2.2 Simulating Work Allocation among a Team

In run-time of the Espresso WfMS, in addition to the team members who
should complete the activity, other team members can execute the activity too.
However this is not always allowed in a real world system. So for the
simulation model, it can be specified, whether just team members who should
complete the activity can execute the activity (see Figure 12-3 in Section
12.1.2).

In the simulation model, the members who execute and/or complete the
activity are determined when the work associated with the activity can be
allocated to the worklist of the team members. The empirical distribution for
allocating a work among the members of a team can be specified to an
Organization Database or a Notes Organization Directory (see Section 12.3.3).
The dummy participant of a team editor can also be allocated with a work (see
Section 12.2 for the description of dummy participant).

**Assumption 12-6. Simulating Workflow Participants among a Team**
If a team editor is assigned to an activity within a process definition and not
all team members should complete the activity, the workflow participants to
execute the activity are uncertain and will be determined when a work
associated with the activity can be allocated to the worklists of the team
members. Suppose \( N \) is the given exact number of members or the number of
given members for the completion of the activity, and \( n \) is the number of team
members participating in the work associated with the process definition.

1° Determine \( m \), the number of workflow participants.
- If just the specified members for activity completion can be workflow
  participants:
  \[ m = N \];
- otherwise (all team members can be workflow participants of the activity):
  \[ m \] is determined by generating a variate governed by \( U[1, n] \)
  distribution, and then is assigned with \( N \) if determined \( m \) is less than \( N \).

2° Determine the workflow participants among the team.
- if given exact number of members for the completion of the activity:
m workflow participants are determined randomly according to the empirical distribution specified to the activity for allocating the work among a team;
• otherwise (given members for the completion of the activity):
  the N given members are included in the determined workflow participants, and other \((m - N)\) workflow participants are determined randomly according to the empirical distribution specified to the activity for allocating work among a team members.

3° add the dummy participant of the team editor to the workflow participants.
• if \(m > n\) (there is not enough members participating in the work associated with the process definition to execute the activity); or
• if some of the determined workflow participants (e.g. in the distribution-specified members or a given member to complete the activity) should be removed since they do not participate in the work associated with the process definition or are not simulated.

According to the assumption, if not all team members should complete an activity within a process definition, the given members for completing the activity will be simulated to execute every instance of the activity; or the simulated number of workflow participants to execute a work associated with the activity will not be less than the given exact number of members for completing the activity, provided that there is enough team members participating in the work associated with the process definition.

### 12.1.1.3 Simulating Worklist of a Workflow Participant

A waiting queue represents the worklist of a workflow in the simulation model and the queuing rule within the simulation model can be specified before a simulation run (see Section 13.2.1.4). A waiting queue is simulated according to the following assumption.

**Assumption 12-7. Simulating Behavior of a Waiting Queue**

1. No polling: there is only one waiting queue formed for a resource and so the sharing of the resource among different waiting queues does not need to consider;
2. No balking: a work cannot refuse to join the waiting queue because of its length, composition, and so on;
3. No reneging: once a work has entered the waiting queue, it must remain in the queue until it has be completed;
4. No jockeying: once having joined a waiting queue, the work can not switch membership to an alternate waiting queue that might, for example, have become short.
Unlike where all team members resolved from a team editor are allocated to a work by the Espresso workflow engine, in the simulation model, a work enters only the waiting queues of the workflow participants who will execute the work.

**Assumption 12-8. Simulating the Waiting Queue of a Team Member**

When allocating a work associated with an activity to the team members that are resolved from a team editor assigned to the activity, the work is added only to the waiting queues of the team members who are determined as workflow participants to execute and/or to complete the work.

From this assumption, for a workflow participant who belongs to a team, the length of the simulated waiting queue may be shorter than the actual length of the worklist in the Application Database, while at run-time a team member must not always do a work allocated in his worklist.

### 12.1.2 Activity Execution Time and Delay Time

Execution time and delay time of an activity affect the duration of an activity. Activity execution time refers to the time taken by the workflow participants to execute the activity. So it contributes to the costs of a WfMS. If an activity is executed by a human workflow participant, for example, the execution time of the activity is used to calculate the costs of human resources.

The elapsed time period that is specified to an activity to just prolong the duration of the activity is called a **delay** of the activity. During delay of an activity, the workflow participants do nothing for the activity. So the delay time is not taken into account in the calculation of the costs of human resources. For example, before a workflow participant can execute an activity, he must call somebody outside the WfMS for some information needed for executing the activity. The time period from the calling till the receipt of the information is the delay for the activity. During the delay of an activity, the workflow participant can not execute the activity, but he can execute other activities in his worklist. **Because** the following factors have been considered in the simulation model for the duration of an activity, they should not be included in the specified delay time of an activity:

- increased execution/occupation time of a resource because it has less weekly participating hours in the work associated with a process definition than the standard weekly working hours;
- waiting for joining;
- waiting for availability of a workflow participant;
- waiting for availability of a start-synchronous material resource; and
- waiting for release of a routing-synchronous material.
That is, the factors that prolong the duration of an activity because of shortage of human/material resources, the imbalance of parallel execution threads of a process instance and partly working time of a resource for a process definition should not be concerned in the delay time specification.

Activity execution time and delay time are specified via the dialogbox of PM as shown in Figure 12-3, where the execution time is called processing time.

Figure 12-3. Activity Resource Definition

The execution time and the delay time are the mean (average time) of a distribution function, if the distribution function is not specified as fixed. Whether an execution time and a delay time are fixed to the given values or are governed by a random distribution function is specified to the configured Notes Organization Directory or Organization Database (see Section 12.3.1).

The standard deviations specified respectively for the execution time and the delay time will be utilized only when the execution/delay time is not fixed and a standard deviation is required for the distribution. For example, an exponential distribution function does not need the standard deviation. If a standard deviation is specified with 0, the determined time will always be the same as the mean, no matter which distribution function is specified.

Assumption 12-9. Distribution of Execution/Delay Time of an Activity
The distribution function of the execution time and the delay time for every activity in an organization is treated as the same. The mean and the standard deviation for the distribution function are individually specified to each activity within a process definition.
The simulation model determines the execution time and the delay time of an activity instance when a work associated with the activity enters the waiting queue of the workflow participants that are resolved from the editor assigned to the activity.

In run-time of a WfMS, it can happen that a workflow participant interrupts the execution of an activity for executing another activity. But in the simulation model, this is not allowed.

Assumption 12-10. Continuous Activity Execution
During simulation, once a workflow participant executes an activity, he continues executing it and can not execute other activities before finishing the execution of the current activity.

In a WfMS, delay may happen during execution of an activity one or more times. But because of the above assumption, the following assumption results.

Assumption 12-11. Simulating Delay of an Activity
When a workflow participant prepares to execute a work waiting in his queue, the delay of the execution will begin, if the determined delay time is larger than zero and it has not elapsed. The delay happens only before execution of the activity. During the delay, the work associated with the activity stays in the waiting queue of the workflow participant and meanwhile he can execute one or more of the other work items (one after another) in the waiting queue.

Because a delayed activity remains in the waiting queue of a workflow participant, the workflow participant can execute the activity at once when the delay time is elapsed and he is idle or becomes no longer busy.

12.1.3 Material Resource Specification

In addition to the workflow participants, some materials may be required for the execution of an activity. Without the presence of the specified materials, the activity can not be undertaken. A material defined for an activity must have the following two features.

- Repetitively utilizable: after being used for execution of one activity, it can be used for execution of another activity.
- Not sharable: during use for execution of one activity, the material can not be used for execution of another activity.

A printer or a fax device is a material that can be used for execution of an activity. In the PM dialog box as shown in Figure 12-3, the materials as well as
the number of units required for the activity execution can be specified. Furthermore, synchronization of each material can be specified through occupation time, synchronous start and synchronous routing:

1. if a material is start-synchronous, a workflow participant of the activity instance can not execute the activity until the material is available to be used for the activity;
2. if a material is routing-synchronous, a work can be routed from the activity only when all routing-synchronous materials used for the activity execution have been released, and of course all workflow participants have completed the activity;
3. if occupation time of a material is not given, it is assumed that the material is used by a workflow participant during the whole time that he executes the activity. Thus the material is both start- and routing-synchronous.

For example, a letter scanned at an activity within a process instance should be routed with the work associated with the process instance to the next activity and so the scanner must be specified as a routing-synchronous material for the activity. A projector must be both start- and routing-synchronous, if it must be used during a presentation, an activity within a process instance, and so the occupation time for the projector should not be specified—the occupation time is the same as the execution time.

If the occupation time of a material is given, it is the mean (average time) of the distribution function for generating material occupation time. The distribution function is specified for an Organization Database or a Notes Organization Directory (see Section 12.3.1). One standard deviation of the distribution is specified to all time-specified materials of an activity.

**Assumption 12-12. Standard Deviation of Material Occupation Time**

The standard deviations of all time-given materials for an activity have the same value.

In the same way a workflow participant can not be interrupted during execution of an activity, the use of a material resource can not be interrupted before it is released from the current work.

**Assumption 12-13. Continuous Occupation of a Material**

Once a material is used by a workflow participant for executing an activity, it can not be used for execution of any other activities before it is released from the occupation.

An activity can not be executed if one of the start-synchronous materials is not available. For a non start-synchronous material, it has no influence to the start of activity execution; but it will postpone the routing of a work associated
with the activity to the next activity if routing-synchronous is specified to the material. Shortages of synchronous material resources will prolong the duration of an activity.

In the Organization Database, the maximum number of units for a material can be defined as one or more. So it is also allowed to let the number of units of the material to be used for the execution of an activity to be greater than one. In the simulation model, it is assumed that all the work items demand the use of a material are waiting in one queue, ignoring whether the maximum number of units of a material is one or multiple.

**Assumption 12-14. Simulating Waiting Queue for Material Occupation**

Only one queue will be simulated for waiting occupation of a material, no matter what the maximum number of units of the material is.

If a material is not simulated, its synchronization definition to an activity will not be used during a simulation run. So the simulation results will not be influenced by the not simulated materials.

### 12.1.4 Resource Costs and Fixed Costs

The most competitive organization is the one which can achieve organization objectives with the best service level or efficiency and the least costs. With help of the simulation study, a system analyst can also improve the allocation of costs between material and human resources.

Human resource costs are calculated upon the execution time of an activity and material resource costs upon occupation time of the materials for execution of the activity. Hourly costs of a human and material resource are defined in the Organization Database configured to the Application Database, in which a process definition is simulated.

Apart from these hourly calculated resource costs, there are some costs calculated upon the number of executed activity instances. These costs are called *fixed costs* of an activity. Fixed costs can contain any kind of costs which are dependent on the execution of an activity. For example, one-time-consumable material (such as a piece of paper) and depreciation charge of shared materials used for the activity execution can be included in the fixed costs. However the costs that have been included in the costs calculated upon hourly costs of human and material resources for the execution of an activity should not be contained in the fixed costs of the activity.

Except for those activities within a process definition executed by Notes agents (IT Resources), all other activities are executed by people. Costs of human resources may take a great portion of the total costs for achieving an organization objective. But reducing human resources may prolong the duration...
of the achievement of objectives and bad service level or low efficiency of the system can let the organization lose customers.

**12.1.5 Simulating Multiple Workflow Participants**

It is known that multiple workflow participants that are resolved from a team editor assigned to an activity can be allocated with a work associated with the activity and execute the work together.

In the Espresso WfMS, an activity instance is represented by a Notes document. If the work associated with the activity instance can be executed by all team members, to avoid replication conflict, the document shared among the team members can be locked when one member is editing it, so that none of the other members can edit it at the same time. In the simulation model however, the lock of the document will not be considered. This does not diverge from the reality, if the time of editing the document representing the activity takes none or little part of the execution time of an activity.

**Assumption 12-15. Simulating Multiple Workflow Participants of an Activity**

If multiple team members as workflow participants are together allocated with a work associated with an activity, at what time and how long a workflow participant executes the activity are independent on one another.

With this assumption, it is not assumed that all the workflow participants of a team must execute the activity at the same time or one after another. But it can happen that some of workflow participants of a team are simulated to execute an activity simultaneously.

For material occupation of multiple workflow participants, the following assumption is applied in the simulation model.

**Assumption 12-16. Simulating Material Occupation by Multiple Participants**

If multiple team members are resolved as workflow participants to execute an activity, each workflow participant will use the materials with the specified number of units for the activity execution.

If an activity is possible to be executed by multiple workflow participants of a team, it is meaningful to specify whether the specified execution time, delay time and material occupation time are dividable by the number of workflow participant as shown in Figure 12-3. If the probability for division of work is specified as 0% to an activity, the times are never dividable for the activity; if 100%, they are always dividable; otherwise, whether the times are divided will be determined according to the given probability (that is, a 0-1 distribution).
**Assumption 12-17. Dividing Work among Multiple Workflow Participants**

Suppose that $\mu$ and $\sigma$ are respectively the specified mean and standard deviation for generating a time for an activity and $n$ is the number of workflow participants to execute together the activity.

- If the work is dividable among multiple workflow participants of the activity:
  - for each of the workflow participants, the mean time and standard deviation are $\mu / n$ and $\sigma / \sqrt{n}$ respectively;
- otherwise (the work is not dividable for the activity):
  - for each of the workflow participants, the mean time and standard deviation are $\mu$ and $\sigma$ respectively.

The formula for dividing time is based upon the central limit theorem.

It is obvious from this assumption that for a time dividable activity, such an activity can be executed through cooperation by multiple people; the more workflow participants work together, the shorter duration is possible for the activity execution; for a not time dividable activity, such as an activity that two people are needed to sign a document, the more workflow participants for the activity, the longer the potential duration.

### 12.2 Dummy Participant

All editors defined in a process definition will be simulated and the states of the editors are graphically displayed in the resource window (see Section 12.4). After a work flows at an activity and it does not need to wait for joining, the work is routed to the editor assigned to the activity and then from the editor to the workflow participants resolved from the editor. The workflow participants can then execute the activity.

It is possible, that an editor assigned to an activity cannot be resolved to a person. In this case, the dummy participant of the editor is used in the simulation model. The dummy participant of an editor is used where one or more of the following conditions apply:

- neither the Organization Database nor the Notes Organization Directory is configured to the Application Database where the process definition is simulated (except editor “Given people”, any other editors defined in the process definition have a dummy participant);
- editor “Anyone” or editor “Computed” is assigned to an activity (the dummy participant stands for the people in the configured Organization Database and/or Notes Organization Directory who have no specification about participation in the work associated with the process definition, and can not be resolved from a team editor either);
• **team** organizational role (i.e. department, workgroup, role or Notes group) not defined in the configured Organization Database or the Notes Organization Directory (the work of the team is undertaken by the dummy participant);
• there is no member in a team organizational role, but all members of the team are specified to complete an activity (the work of the team is undertaken by the dummy participant);
• a member of a team organizational role does not participate in the work associated with the process definition, if all members of the team should complete an activity within the process definition;
• the number of members resolved from a team editor is less than the specified number of members to complete the activity;
• given members to complete an activity are excluded from the simulation;
• the human resources participating in a work associated with the process definition are excluded from the simulation; etc.

The dummy participant of an editor assigned to an activity within a simulated process definition is named by “(Dummy: <Team Name>)”, such as “(Dummy: Anyone)”, appending with a combination of ReplicaIDs of an Organization Database, a Notes Organization Directory, and/or an Application Database, which are configured to the process definition and related to the editor. For example, the Replica ID of the Application Database, in which a process definition is simulated, is included in the name of the dummy participant for editor “Computed”, because the Notes field specified for the editor is defined (in a document saved) in the database and so the database is related to the editor.

Dummy participants make the simulation states and results intact, especially when not all workflow participants are required to be simulated. By observing the dynamic state data or studying statistic data of dummy participants, the system analyst can also discover integrity errors within the process definitions.

**Assumption 12-18. Dummy Participant of an Editor**

• The work associated with an activity will be resolved wholly or partly to the dummy participant of the editor assigned to the activity if the editor cannot be resolved to the specified workflow participants or some workflow participants are not simulated.
• Dummy participants have no costs, i.e. hourly cost of a dummy participant is zero.
• A dummy participant is sharable and so he can execute an activity as soon as the work associated with the activity is allocated to him.
• There is no specification of weekly participating hours for a dummy participant—it is the same as the standard weekly working hours.

See Section 12.3.2 for the settings of weekly participating hours of a resource and standard weekly working hours for a simulation run.
Except those mentioned in the assumption, a dummy participant executes an activity in the same way as a usual workflow participant, such as execution time determination, material occupation, etc. Because the dummy participant of an editor is unlimitedly sharable, there is no waiting queue for it. When a dummy participant is allocated with a work, he can execute the work at once. No weekly participating hours is specified for a dummy participant, since the time taken by a dummy participant is treated as that required for the activity execution so that the related statistical data can be easily analyzed.

12.3 Organizational Settings

Human and material resources are defined in the Organization Databases and Notes Organization Directories. The simulation settings specified to the databases are organizational settings. They can be categorized into four groups:

- distribution function settings,
- participant settings,
- team work assignment settings, and
- public holiday settings

The organizational settings can be either process-dependent or process-independent. Process-independent organizational settings are used as default settings for a simulated process definition configured with the Organization Database or the Notes Organization Directory if there is no corresponding process-dependent organizational settings for the process definition.

12.3.1 Distribution Function Settings

The distribution functions are specified for the simulation model to generate activity execution times as well as delay times, work routing times and material occupation times. Parameters of the distribution functions (such as mean and standard deviation) are specified with the definitions of an activity or a link. The distribution functions can be either normal, uniform, exponential or gamma distribution, as shown in Figure 12-4.

The curve of a distribution function with an example of mean value will be plotted by PM during the selection of a function. It varies with the change of the standard deviation and presents as an example how the standard deviation influences the distribution function. The values of the left and right margins can be adjusted for altering the display range of the function curve. The shaded area presents the distribution of the probability of a random variable over its
range—the higher the curve on a value, the more possible that the value will be taken by the random variable following the distribution function.

In a real world business process, the execution time, delay time, routing time, and occupation time will vary with various unexpected factors and the distribution function for generating them will not be set as fixed to the given mean. This option for specifying fixed time is particularly useful when the system analyst want to test and validate the simulation model, to comprehend the simulated system, or to compare different decision policies.

Assumption 12-19. Effect of Zero Standard Deviation
If the standard deviation is specified with zero, the generated time will always be the same as the mean, no matter which distribution function the time follows.

12.3.2 Participant Settings

A lot of resources may be involved in a simulated WfMS, especially in such a case that an activity within a process definition is assigned to editor “Anyone”, editor “Computed”, or a large team (such as editor “Role”, “Department”, “Workgroup”, or “Group”) with many members. There are some disadvantages to simulate many resources in a simulation run:
it takes a lot of time to allocate a work among a big team;
for each simulated resource, memory as well as time is needed to keep
and refresh state and statistic data; and
it is complicated to analyze the simulation results and to make decisions
upon them.

The dialogbox as shown in Figure 12-5 is used to specify weekly hours that

![Figure 12-5. Participant Specification](image)

a human or material resource can participate in a work associated with a
process definition and whether the resource will be simulated or not. If the
value of weekly participating hours of a resource is set as zero, the resource
does not participate in a work associated with the process definition.

**Assumption 12-20. Resource Not Participating in a Process Work**
A human or material resource with zero weekly participating hours for a
process definition does not participate in the work associated with the process
definition.

A material that has been specified in a process definition can not have zero
weekly participating hours for the process definition.
If a member of a team editor has no specification in the Simulation Database
about the participation in the work associated with a process definition, the
team member will be simulated and his weekly participating hours is the same as the standard weekly working hours.

Only the resources participating in a work associated with a process definition will be included in the simulation model. If a team editor is assigned to an activity within a process definition and all team members should complete the activity, the work belonging to a member who is not participated in the work associated with the process definition will be allocated to the dummy participant of the editor. Suppose a role has ten members but only two of them participate in the work associated with a process definition, during simulating the process definition, a work assigned to the role will be allocated to the two members as well as the dummy participant of the editor “Role”, if all members of the role should complete the activity.

The non-zero weekly participating hours of a resource works with the standard weekly working hours.

**Assumption 12-21. Work Calendar**
The standard weekly working hours is specified in the calendar settings of weekly working days and daily working hours in the user preferences of PM.

According to the assumption, it is assumed that all organizations involved in a simulation run have the same standard weekly working hours.

If weekly participating hours of a simulated resource for a process definition is less than the standard weekly working hours (the resource is part-time available or employed for the business processes in accordance with the process definition), the time taken by the person for executing an activity or by a material for the occupation will be longer than the usual time; On the other hand, if the weekly participating hours of a person, for example, is larger than the standard weekly working hours, he works overtime and so needs less time to execute an activity. Suppose the standard weekly working hours are 40, a person participates in the work associated with a process definition 20 hours a week, and an activity within the process definition needs 100 hours for the execution, the duration will be on average 200 hours.

**Assumption 12-22. Effect of Participant Settings**
The duration caused by a human or material resource is calculated by $t (S / P)$

Here $P > 0$ is the weekly participating hours of the resource, $S$ is the standard weekly working hours, and $t$ is the given time for executing an activity by the person or used material.

A team with a defined parameter or without a parameter, such as a workgroup, participates in the work associated with a process definition if it is assigned to an activity within the process definition. But if a team, such as a role, has multiple parameters and the parameter of that team is dynamically determined at run-time, some teams with specific parameters may not
participate in the work associated with a process definition in a real world system. This can be set in the dialog box as shown in Figure 12-6.

12.3.3 Team Work Assignment Settings

The empirical distribution for allocating a work among members of a team can be specified in the dialog box as shown in Figure 12-7. Once it is associated with a process definition, it is activity-dependent.

Not all team members must be specified in the empirical distribution. The probability for others is the sum of the probabilities of not specified team members who participate in the work associated with a process definition. If it is not zero, a work can be assigned to a member who is not specified in the distribution but participates in the work associated with a process definition. In this case, the determination of the member from the others will be governed either by the uniform distribution function or by the rule to assign the work to the member with shortest queue, as specified by the analyst.

In the example in Figure 12-7, department “Credit Proof” has six members and it is assumed they all participate in the work associated with process definition “Loan”. When the team gets a work associated with activity “Evaluate” and one member should execute it, the probability for Jane Colliver, Shirley Hindley and the other four members to execute the work are 30.0%,
20.0% and 50.0% respectively. If none of the two specified members in the distribution is allocated with the work, the other four members, whose probability is not individually given in the distribution, have the same probability (i.e. 12.5%) to execute the activity.

According to the work allocation distribution for an activity within a process definition, a team member, who is specified in the distribution, or who participates in a work associated with the process definition and the probability for others is larger than zero, will be allocated with a work associated with the activity. If the member is not simulated, his work will be executed by the dummy participant of the editor assigned to the activity.

If a team has several parameters and the parameter for determining workflow participants is given at run-time, the allocation distribution for determining the parameter of the team can also be specified as shown in Figure 12-8. In the allocation distribution, all the parameters of a team will be automatically filled in the list.
12.3.4 Public Holiday Settings

Public holidays of an organization are specified in the dialog box as shown in Figure 12-9. The public holidays of the database are initialized with the default public holidays saved in a Notes database, such as a calendar database of an organization, where each public holiday is defined in a Notes document and where the documents of the public holidays are listed in a Notes view (see Figure 12-10).

The public holidays like weekends will be excluded when the simulation model transforms the value of the clock into a date and from a large time unit (such as week) to a small unit (such as hour). They are also excluded during date specification when increasing or decreasing the value of a date.

The absolute simulation time represented by the clock is sufficient for analysis of the system performance and resource bottlenecks in a WfMS. But when the analyst wants the state and statistic data to be related with the date during a simulation run, he should specify the simulation beginning date. Thus the absolute time can be transformed to a date based on the settings of weekly working days (e.g. 5) and daily working hours (e.g. 8), excluding Saturday, Sunday and common public holidays as well.
If both an Organization Database and a Notes Organization Directory are configured to an Application Database where a process definition will be simulated, only the common public holidays which are both defined in the Organization Database and in the Notes Organization Directory are considered. For example, if March 8 is a public holiday in the Organization Database, but not in the Notes Organization Directory, the day is not treated as a public holiday of the simulated WfMS.

**Figure 12-9.** Public Holiday Specification

**Figure 12-10.** Default Public Holiday Specification
**Assumption 12-23.** Common Public Holidays
If one or several process definitions will be simulated in diverse Application Databases, only the holidays included in all configured Organization Databases and Notes Organization Directories will be considered in transforming the absolute simulation time to a date.

### 12.4 Graphic States of Resources

Human resources, material resources and Notes agents can be assigned to a simulated process definition. At run-time, the names of human resources will be obtained from either a Notes Organization Directory or an Organization Database, or both; material resources will be retrieved from the Organization Database; and Notes agents are programmed in the Application Database. Because some resources defined in the databases are not assigned in a process definition, they will not be considered in the simulation study.

The system analyst can specify whether a human or material resource will have the opportunity to participate in the work associated with a process definition. Resources participating in a work associated with the process definition should have a non-zero weekly participating hours. Only the resources participating in the work associated with the process definition can be simulated. The state of a simulated resource can be displayed in the resource window of PM. The relationship between specified, participated, simulated and displayed resources is illustrated in Figure 12-11.

![Figure 12-11. Simulated Resources](image)

All editors assigned to one or several activities within the analyzed process definitions will be simulated. A Notes agent assigned directly as a workflow participant in the process definitions will be simulated too.

Dynamic states of an editor and a simulated resource are graphically displayed with the icons and data as described in Figure 12-12. They can be categorized into three groups.
Material resource: the icon with two intersected rulers represents a material resource. Name of the material, the maximum number of units of the material, and current state of the queue length as well as available units are displayed beside the material icon.

Workflow participant (human resource or Notes agent): for each workflow participant, one of two icons is used to represent whether he is busy or idle. The icon of a walking person indicates that a workflow participant is executing an activity and he is busy; the icon of a standing person indicates that a workflow participant is idle. Beside either the idle or busy icon of a workflow participant, a graphic queue consisting of a series of document icons may appear. If a workflow participant is busy, the number of the process instance associated with the work he is undertaking and the time needed by him to execute the work are displayed on the icon of the workflow participant.

Editor: the icon of a pair of people represents the assigned editor. Editor name, current number of running activity instances assigned to the editor, and the number of the last arriving or routed (to next editor, to workflow participants, or back from a workflow participant) process instance are displayed beside the editor icon. In Figure 12-12 for example, process instance “1/31.”, i.e. process 31 within the first process definition in PM, is the last one to be routed.

In the Espresso run-time Application Database, there is a Notes view serving as the worklist for every human workflow participant of the WfMS. A workflow participant can execute the activities allocated under his name. The worklist is represented by a queue in the simulation model of PM. If the value of a queue length is not displayed at the end of the graphical queue, the number of document icons building the queue is equal to the queue length. The maximum number of document icons displaying in a queue (here four) can be
changed before or during a simulation run (see Section 13.2.2). The value of queue length (here 7 for workflow participant Jill Dando) is displayed only when the queue length is larger than the maximum number of displayed document icons.

A document icon in the resource window represents a work associated with a process instance in accordance with a simulated process definition. The number of the process instance is displayed on it. Also a document icon can be used to build up a queue, it is also utilized to animate a flowing work. During simulation, a document icon will flow in one of three different directions:

- from one editor icon to another editor icon, when a work is routed from one activity to another;
- from the icon of an editor to the icons of the workflow participants resolved from the editor, when allocating a work assigned to the editor; and
- from the icon of a workflow participant back to the icon of the editor from which the workflow participant was resolved, when the work is completed by the workflow participant.

A line with a document icon on it presents graphically the direction in which a work flows. If a work flows from an editor to multiple workflow participants or several next editors, for each simulated workflow participant or each next editor, such a line will be displayed simultaneously. For example, in Figure 12-13, the work associated with process 135 and assigned to editor “Department” of “Personal Creditworthiness Proof” is just being allocated to two workflow participants of Suellyn Hayes and Peter Marley; one work associated with process 134 is flowing from editor “Role” of “Accountant” to editor “Department” of “Customer”, another associated with process 135 is being

![Dynamic Resource Window](image)

Figure 12-13. Dynamic Resource Window
When a work is routed to an activity, the editor assigned to execute the activity receives the work from the editor of one predecessor activity. If the current activity is a join activity, the work cannot be allocated to workflow participants until there are no more parallel work items of the same process instance with the potential to flow to the join activity.

When a work flows to a workflow participant, it will enter the waiting queue and then wait there if the workflow participant is busy. For the example in Figure 12-13, the work associated with process 135 must wait in the queue of Peter Marley because he is busy with the work associated with process 134. If the workflow participant can execute the work immediately, the icon representing the workflow participant will be switched from idle to busy and the number of the process instance and execution time of the work will appear meanwhile. After he completes the work, he can execute another work waiting in his queue. The just completed work by the workflow participant is sent back to the editor icon, from where the work came.

A work can be routed from an activity to the next only when all workflow participants allocated with the work have completed it and all routing-synchronous materials used for the execution of the activity have been released.

The dynamic state data, such as the execution time of current work and queue lengths, can be seen as an auxiliary tool to detect potential resource bottleneck. For example, if a lot of running work items are assigned to an editor, the editor could be a potential bottleneck for the WfMS.

### 12.5 Data Structure of a Queue

For each human workflow participant, the waiting queue can be graphically displayed. The queue records combining a waiting queue are connected with two pointers for each queue record as shown in Figure 12-14—one is pointed to the next record and another to the previous record. For each workflow participant that is kept in a participant record, another two pointers are used to connect respectively head and tail of his queue. A queue record, which keeps
the data of a work associated with an activity, contains the execution time of the activity. In this example in Figure 12-14, there are two work items waiting in the queue, one needs 2 minutes to execute and the other 5. With this data structure, it is easy to handle a queue according to different queuing rules.

Now suppose a new work with execution time 3 minutes will be added into the queue in Figure 12-14. If the queuing rule is first-in-first-out, the queue becomes that as shown in Figure 12-15; if the work is added in the rule of last-in-first-out, the result will be that as shown in Figure 12-16; and if the work is added so that the queue is ordered in respect to execution time, the result is then as that shown in Figure 12-17.
12.5.1 Algorithm for Adding Work to Queue

This procedure is called when processing work allocation event (see Section 13.4) to add a work in the queue of a workflow participant.

Hypothesis
Here participant record \( p \) and queue record \( i \) are parameters of the procedure. The work in queue record \( i \) is newly created and it will be added to the queue of the workflow participant kept in participant record \( p \).

Suppose \( \text{QueueLength}(p), \text{Head}(p) \) and \( \text{Tail}(p) \) represent respectively queue length, queue head pointer and queue tail pointer of the workflow participant; \( \text{ExecutionTime}(j), \text{Next}(j) \) and \( \text{Previous}(j) \) stand for respectively the execution time, next queue record pointer and previous queue record pointer of a work kept in queue record \( j \).

Principle
A work in queue record \( i \) is added in the queue of the workflow participant in participant record \( p \) according to given queuing rule (see Figure 12-14, Figure 12-15, Figure 12-16 and Figure 12-17).

Procedure \((p, i)\)

Step 1: \( \text{ExecutionTime}(i) \leftarrow \text{generating execution time for the work in queue record } i \);

Step 2: if \( \text{QueueLength}(p) > 0 \) (a queue of the workflow participant in participant record \( p \) is already existing), go to Step 4;

Step 3: (to build up a new queue for the workflow participant in participant record \( p \)) \( \text{Next}(i) \leftarrow 0; \text{Previous}(i) \leftarrow 0; \text{Head}(p) \leftarrow i; \text{Tail}(p) \leftarrow i; \) go to Step 12;

Step 4: if the queuing rule is first-in-first-out, go to Step 10;

Step 5: if the queuing rule is last-in-first-out, go to Step 11;

Step 6: (the queue is ordered according to the least execution time) if \( \text{ExecutionTime}(i) < \text{ExecutionTime}(\text{Head}(p)) \), go to Step 11;

Step 7: if \( \text{ExecutionTime}(i) \geq \text{ExecutionTime}(\text{Tail}(p)) \), go to Step 10;

Step 8: let \( j \) be a queue record in the queue with

\[ \text{ExecutionTime}(i) \geq \text{ExecutionTime}(j) \text{ and } \text{ExecutionTime}(i) < \text{ExecutionTime}(\text{Next}(j)); \]

Step 9: (insert queue record \( i \) after record \( j \)) \( \text{Next}(i) \leftarrow \text{Next}(j); \)

\( \text{Previous}(\text{Next}(j)) \leftarrow i; \text{Previous}(i) \leftarrow j; \text{Next}(j) \leftarrow i; \) go to Step 12;

Step 10: (append queue record \( i \) to the tail of the queue) \( \text{Next}(i) \leftarrow 0; \)

\( \text{Previous}(i) \leftarrow \text{Tail}(p); \text{Next}(\text{Tail}(p)) \leftarrow i; \text{Tail}(p) \leftarrow i; \) go to Step 12;

Step 11: (insert queue record \( i \) to the head of the queue) \( \text{Next}(i) \leftarrow \text{Head}(p); \)

\( \text{Previous}(i) \leftarrow 0; \text{Previous}(\text{Head}(p)) \leftarrow i; \text{Head}(p) \leftarrow i; \)

Step 12: accumulate integrated value of queue length of the workflow participant in record \( p \);

Step 13: \( \text{QueueLength}(p) \leftarrow \text{QueueLength}(p) + 1; \) stop.
12.5.2 Algorithm for Removing Work from Queue

This procedure is called when processing the participant depart event or the material release event (see Section 13.4) to remove the queue record which keeps the first executable work from the queue. It is called before a workflow participant begins executing an activity.

**Hypothesis**
Here participant record \( p \) and queue record \( i \) are parameters of the procedure. The work in queue record \( i \) is the first work in the queue that can be executed (all required material resources are available and delay is no more required).

Suppose \( \text{QueueLength}(p) \), \( \text{Head}(p) \) and \( \text{Tail}(p) \) represent the queue length, the queue head pointer and the queue tail pointer of the workflow participant in participant record \( p \); \( \text{Next}(j) \) and \( \text{Previous}(j) \) represent the next queue record pointer and the previous queue record pointer of a work kept in queue record \( j \) respectively.

**Principle**
Queue record \( i \) of the workflow participant in record \( p \) will be removed from the queue of the workflow participant.

**Procedure** \((p, i)\)

Step 1: if \( i = \text{Head}(p) \), go to Step 4;
Step 2: if \( i = \text{Tail}(p) \), go to Step 5;
Step 3: (remove queue record \( i \) from the middle of the queue) \( \text{Next} (\text{Previous}(i)) \leftarrow \text{Next}(i); \text{Previous} (\text{Next}(i)) \leftarrow \text{Previous}(i) \); go to Step 6;
Step 4: (remove queue record \( i \) from the head of the queue) \( \text{Head}(p) \leftarrow \text{Next}(i); \text{Previous} (\text{Next}(i)) \leftarrow 0 \); go to Step 6;
Step 5: (remove queue record \( i \) from the tail of the queue) \( \text{Tail}(p) \leftarrow \text{Previous}(i); \text{Next} (\text{Previous}(i)) \leftarrow 0 \);
Step 6: accumulate waiting time of the queue of the workflow participant in record \( p \);
Step 7: accumulate integrated value of queue length of the workflow participant in record \( p \);
Step 8: \( \text{QueueLength}(p) \leftarrow \text{QueueLength}(p) - 1 \); stop.

12.6 Conclusion

Many input variables concerning resources must be specified for the simulation model.

The resources and costs required for the execution of an activity within a process definition is specified with the definition of an activity. At run-time,
workflow participants of an activity are resolved to people or a Notes agent from an editor assigned to the activity. Unlike the Espresso workflow engine which allocates a work to all members of a team, the simulation model allocates the work only to the workflow participants of the team.

The given execution time of an activity is used by the simulation model to generate the time taken by the workflow participants for the execution of the activity. The costs of human workflow participants are calculated upon the execution time.

The material resources used for execution of an activity can be specified and simulated. The costs of materials are calculated according to the occupation time of a material. The start-synchronous or routing-synchronous materials may postpone starting execution of an activity or routing a completed work from the activity.

The specified delay of an activity is the time period that should be elapsed before a workflow participant undertakes the activity. The duration caused by shortage of resources and unbalanced parallel execution threads of a process instance should not be considered in the specified delay time of an activity.

Costs of human and material resources are calculated from the hourly costs defined in the Organization Databases. Other costs are calculated from the fixed costs specified to an activity.

Time distribution functions, weekly participating hours of a resource, the rule to allocate a work among a team, and public holidays are specified in an Organization Database or a Notes Organization Directory.

The dummy participant of an editor will be allocated to a work associated with a process definition if a workflow participant resolved from the editor does not participate in the work associated with the process definition or is not simulated.

The dynamic states of simulated resources and routed work items can be graphically displayed in the resource window during simulation. The queue of a human workflow participant will be graphically simulated and so the data structure of the queue was discussed in the chapter.
13 OVERALL SIMULATION MODEL

The simulation model implemented in PM allows process definitions to be graphically simulated or animated. A process definition can be simulated in one or multiple Application Databases. Simulated resources are retrieved from Organization Databases and/or Notes Organization Directories that are configured to the Application Databases.

Before a simulation run, values of input variables can be specified via different simulation settings. During simulation, dynamic state variables of the simulated WfMS can be graphically displayed on the process maps and in the resource window. After simulation, statistical simulation reports are created and can be presented in charts or tables and then saved and/or printed. Simulation reports summarize the simulated system performance and can be utilized for analyzing bottlenecks of a WfMS in order to improve system performance.

13.1 Experimental Modes

PM can simulate work items associated with the process instances in a WfMS and let them flow from activity to activity, from activity to workflow participants and from workflow participant to activity, in accordance with the process definitions. The following three alternative simulation modes can be experimented in PM.

- **Animate one process instance:** one process instance is created in an Application Database at a specified start activity of a process definition. The process instance can be terminated at a specified activity reachable from the start activity. During animation, the analyst should interact in decision-making;
- **Simulate one process definition:** any number of process instances in accordance with a process definition can be simulated in one or multiple Application Databases. No user interaction is allowed during simulation;
- **Simulate all opened process definitions:** process instances in accordance with different process definitions will be simulated in diverse Application Databases. No interaction is allowed.

In all the experiment modes, routed work items associated with the process definitions can be animated and some state/statistic variables can be graphically presented on the process maps and in a resource window.
13.1.1 Animation

Animation is used to test a process definition step by step and to see how a process definition will be enacted in an Espresso Application Database. During animation, the system analyst should interact in the experiment to make various decisions.

- Process creation decision: determine a workflow participant for creating the process instance at a start activity, if multiple workflow participants can be resolved from the editor assigned to the activity (see Figure 13-1).
- Member decision: choose members among a team for executing and/or completing an activity, if not all members of the team should complete the activity (see Figure 13-2).
- Variate decision: choose the value of a variable defined in a “Condition” in an outgoing link of the activity which is just completed (see Figure 13-3).
- Branching decision: choose “Multiple Choice” outgoing links and/or an “Exclusive Choice” outgoing link of an activity, over which the work associated with the activity can flow further (see Figure 13-4).
- Parameter decision: choose parameter of a role if it is specified as being dynamically determined at run-time.

After animation, a process protocol from the activity where the process instance is created till the last completed activity will always be generated. The statistical simulation reports will also be created although just one process instance is simulated.

![Figure 13-1. Choose Creator of a Process Instance](image)
Figure 13-2. Choose Workflow Participants of an Activity

Figure 13-3. Choose Variate of a Variable
The usage of animation is to:

- estimate costs and duration of a process instance from a start activity to an end activity of the process definition;
- analyze a join activity to see whether the duration of different execution threads of a process instance are balanced (detect critical paths);
- test whether a process definition will act as expected, especially where parallel work items are joined, how join priorities influence the release of deadlocks, etc.;
- experiment with a process definition and see what kind of decisions should be made by workflow participants at run-time, and how different decisions will influence the execution threads of a process instance in accordance with the process definition;
- evaluate and learn the simulation results, since animation is the simplest simulation mode and so the statistical results can be comprehended most easily; or
- forecast how different operating policies will affect the business processes in accordance with the process definition.
13.1.2 Simulation

One or all opened process definitions in PM can be simulated simultaneously in different Application Databases, like an Espresso WfMS can be constructed in the real world. Simulation results can be used for analyzing bottlenecks, costs, system performance, etc. The graphic simulation procedure helps the analyst to observe how different resources, processes and activities will interact in a WfMS.

During simulation the analyst can not interact to make decisions as in animation. All these decisions are made stochastically by the simulation model according to the corresponding distribution functions.

Simply to simulate one process definition in one Application Database can help the analyst to estimate system parameters for the process definition, especially the distribution functions for some random input variables.

13.2 Simulation Settings

In PM, various data obtained in different ways are used by the simulation model.

- Specification of activities: editor, execution time, delay time, fixed costs, material resources (units, time and synchronization) and escalation data (maximum duration, execution time and number of running instances).
- Specification of links: routing option and time.
- Process settings: life period, process intercreation time, routing probabilities of “Multiple Choice” links, routing distribution of “Exclusive Choice” links, and variate distributions of variables defined
- Organizational model: definitions of organizational roles and material resources; and hourly costs of human/material resources.
- Resource settings: time distribution functions for execution/delay of activities, routing of work and use of materials respectively; participating and simulating resources; work allocation distribution among members of a team; and public holidays.
- Experimental settings: the unit of the clock, simulation run length, protocol generating intervals, and the Application Databases as well as configured Organization Databases and/or Notes Organization Directories for each simulated process definition (see Section 13.2.1).
- Simulation view settings: for displaying graphically the simulated states of a WfMS, animating of routing work items, making sounds and displaying some dynamic messages during a simulation run (see Section 13.2.2).
Specification of activities and links within a process definition are saved with the process definition in a Process Database. Organizational models are defined in Organization Databases and/or Notes Organization Directories. Process settings, resource settings and experimental settings are stored in the Simulation Database. Simulation view settings are user preference settings for a simulation run. They can be modified during simulation and so are not saved in the Simulation Database.

Apart from simulation view settings and some experimental settings, all others are input variables of the simulation model. They are determined by the factors of operating policy, technical support, human and material resources, work atmosphere/regulation of an organization, marketing, etc. and should not be given willfully. The input data can be collected, for example, via statistical analysis of history data, or be forecast by experts. Incorrect input data will cause the simulation results to be worthless and unusable.

13.2.1 Experimental Settings

The experimental settings are prepared for a run of a simulation experiment. It should be specified before an animation/simulation run. Experimental settings include the time unit of the clock, simulation beginning/ending conditions, interval to generate process protocol at specified activities, databases configured to the simulated process definitions, and queuing rules of simulated worklists of workflow participants.

The experimental settings can be saved as a scenario in order to use them later for further simulation runs of the same scenario. Different simulation runs of the same scenario may generate different simulation results, because a WfMS is usually a stochastic system. For the evaluation of the simulation results, multiple runs of a scenario are necessary.

13.2.1.1 Simulation Beginning/Ending Conditions

Simulation beginning/ending conditions determine the simulation run length. For a simulation run, the time period when a process instance can be created, or the maximum number of process instances to be created, are specified in the dialog box as shown in Figure 13-5.

The time period specifications are applied to the clock. At the beginning of a simulation run, the clock is assigned with value zero.

If the simulation beginning date is specified, a simulation run is assumed to begin on the date corresponding to the clock value zero. So the value of the clock can be transformed to a date according to the simulation beginning date and the standard weekly working hours, excluding Saturday, Sunday and
PART THREE: WFMS SIMULATION

common public holidays. The clock can be displayed in date form only when the simulation beginning date is given.

Three conditions can be specified for the creation of process instances in accordance with a process definition. A process instance can be created, if

- the clock does not exceed the given maximum time;
- the date transformed from the clock does not go beyond the given simulation ending date; and
- the number of created process instances does not exceed the given maximum number of process instances.

That is, no more new process instances will be created in the simulated WfMS if one of the stipulated conditions arises.

If the maximum simulation time or the simulation ending date is given, the analyst can specify whether to simulate all running process instances till their termination when the time limit arises. If so, the total simulated time may extend beyond the given maximum time.

The eventlist of the simulation model becomes empty after all running process instances in the simulated WfMS are finished and thus the simulation run terminates.

In addition to above specified conditions, the creation of process instances in accordance with a process definition is simulated within the defined life period of the process definition. For example, suppose a process definition has life ending date 12/31/1999. According to the above experimental settings in Figure 13-5.

Figure 13-5. Simulation Beginning/Ending Settings
13-5 (the simulation beginning date is 01/01/2000), no process instance in accordance with the process definition will be created and simulated.

Life ending dates and the simulation ending date work with the simulation beginning date. So if a simulated process definition has a life beginning date or a life ending date, the simulation beginning date must be given.

If no simulation beginning/ending condition is given and not all simulated process definitions have life period specifications, the simulation run can only be manually interrupted by the system analyst.

### 13.2.1.2 Process Protocol Settings

Process protocol settings as shown in Figure 13-6 are to specify at which activity and in which instance interval in an Application Database to generate the process protocols of process instances in accordance with a process definition. The specification is related to an Application Database where the process definition is simulated.

![Figure 13-6. Process Protocol Settings](image)

A process protocol (see Section 13.3.1) presents an execution thread of a process instance from the start activity where the process instance is created up to the specified activity. In this example, a process protocol of activity sequence to activity “Implement loan” will be generated, when every second work associated with the activity is completed.
During animation, a process protocol will be automatically generated after an end activity of the process definition or the specified animation ending activity is completed.

Generated process protocols are saved in the Simulation Database and can be displayed in table form during simulation and animation.

### 13.2.1.3 Database Settings

The Application Databases in which a process definition will be simulated are specified here. The configured Organization Database and/or Notes Organization Directory to an Application Database are utilized for retrieving simulated human and material resources. Figure 13-7 presents how the settings are defined in PM.

A process definition can be simultaneously simulated in several Application Databases. In this example, process definition “Loan” will be simulated concurrently under two Application Databases with the same or different configurations of Organization Databases and/or Notes Organization Directories.
13.2.1.4 Queuing Rule for Worklist

The principle of sorting the worklist of a workflow participant in an Application Database instructs or conducts the workflow participant through the work items allocated in his worklist—usually a workflow participant pays most attention to the first entry in the worklist. So different sorting principles can result in different system performances. Settings of queuing rule for worklists as shown in Figure 13-8 let the system analyst decide which of the following principles is the best for sorting the worklists in an Application Database:

- in the order of arrival time (first-in-first-out); or
- in the order of recent arrival (last-in-first-out or first-in-last-out); or
- in the order of shortest execution time.

**Assumption 13-1. Queuing Rule**
A workflow participant always undertakes the first executable work (material available and no delay required) in his waiting queue.
13.2.2 Simulation View Settings

The simulation view settings are user preference settings that can be modified before and during a simulation run. Simulation view settings specified in the dialog box as shown in Figure 13-9 includes:

- the speed for animating routed work items,
- steps for animating a routed work along a link,
- maximum number of document icons displayed in a waiting queue of a workflow participant,
- whether to show the resource window,
- whether to display the clock in date form,
- whether to pop up the descriptions of dynamic displayed data,
- whether to prompt messages about a stopped work,
- whether to prompt generated protocols,
- the sound for various situations (see Figure 13-10),
- which process map windows are displayed (see Figure 13-11), and
- which editors, human and material resources will be illustrated (similar to Figure 13-11.), if resource window is shown.

![Figure 13-9. Simulation View Overall Settings](image-url)
The simulated process instances represent the business processes running in an organization for example offering services or products. For such a WfMS system, the most important system performances are:
• total completed process instances corresponding to total costs—profit of an organization; and
• number of running process instances in a system as well as duration of a process instance—congestion and efficiency of the system and customer satisfaction.

The system performance criteria generated after simulating the Espresso WfMS are categorized in the four statistical reports:

• summary report,
• activity report,
• resource report, and
• work allocation report.

A dynamic process protocol generated during simulation presents how an execution thread of a process instance is performed from the start activity where the process instance was created up to the specified activity.

The statistical results of a simulation run as well as the generated process protocols can be saved in the Simulation Database. The scenario must be saved before the reports are saved.

13.3.1 Process Protocol

Process protocols can be generated automatically (during animation) or as scheduled (during simulation). A process protocol as shown in Figure 13-12 is a dynamic report about an execution thread of a process instance. It presents when the process instance is created, who created the process instance, through which activities the work associated with the process instance has flowed to the current activity (the last activity in the table), duration of the execution thread from the first activity up to the current activity (= depart time of the activity), and the critical path (combined by activities marked with “*” or “**”). For each activity, the following data are displayed in the protocol:

• whether the activity is on the critical path of the execution thread of the process instance (marked with “*”), and whether it must be undertaken for any process instance in accordance with the process definition from the start activity up to the current activity (marked with “**”);
• routing message (when the work departed from a predecessor and when it arrived at the activity);
• when the first (parallel) work from all predecessors arrived;
• duration of the work at the activity;
13 OVERALL SIMULATION MODEL

- total execution time of the work by all workflow participants of a team assigned to the activity;
- who executed the work and how much time was taken; and
- when the work is routed out from the activity.

The parallel execution threads within the process protocol are shown in the table too. The data in the column “Previous Task(s)/Depart->Arrival Time” is a set of previous activities with corresponding depart times from the previous activity and arrival times to the activity. In this protocol example, activities “Check asset valuations” and “Check personal creditworthiness” were executed in parallel and were joined at activity “Approve credit”. Marked links on the process map in Figure 13-13 present graphically how work items associated with the process definition flow from activity to activity. (Marked activities on the process map combine the critical path of the execution thread.)

In this example, the process definition is simulated simultaneously in two Application Databases. The data beside each activity corresponds to the two databases respectively (see Figure 11-5 for the descriptions of the data). From Figure 13-12, it is known that process 4 has just completed at activity “Implement loan” in the Application Database with the Replica ID “C1256768:00377BA7”. So the first part of data beside each activity corresponds to this application database.
Duration of a work at an activity is the time period between the arrival time of the first parallel work and the depart time of the (joined) work from the activity. During this time period, multiple workflow participants of a team may execute the activity simultaneously. Duration includes also specified delay time of the activity and times waiting for joining, for resource use, as well as for resource releasing. The waiting times are reported in the statistical activity report (see Section 13.3.3). The join-waiting time can also be computed from the data in column “Previous Task(s)/Depart→Arrival Time” (the last arrived subtracts the earliest arrived). If duration minus join-waiting time is much larger than the longest execution time plus delay time of all the workflow participants, some workflow participants can not execute the activity soon after the work is allocated in their worklist—they may have too many activities to do, or there is a shortage of the synchronous materials.

Long duration of a process instance causes a low service level or inefficiency of a WfMS. Process protocols can be used as a tool to improve performance of a system by analyzing critical path and then balancing parallel execution threads. A critical path is the execution thread of activities contributing to the duration of a process instance. If a process protocol contains joined parallel execution threads, the time waiting for joining at a join activity prolongs the duration of the process instance and should be reduced. To let the activity on the critical path depart the activity as soon as possible can reduce the overall duration of the process instance. Prolonging the duration of one activity involved in the critical path will lengthen the overall duration of a process instance.

Figure 13-13. Process Definition and Process Protocol
Process protocols are saved in the Simulation Database with other simulation results. They can be displayed during simulation.

### 13.3.2 Simulation Summary Report

The simulation summary report as shown in Figure 13-14 includes general results of a simulation run. They are:

- name of the relevant simulation scenario (saved experimental settings),
- generating time of the simulation reports,
- total created and completed process instances,
- simulated time period,
- start and end date of the simulation run,
- total costs of the simulated system, and
- the list of simulated process definitions.

The example in Figure 13-14 presents the experimental results after simulating process definitions in Figure 1-3, Figure 1-4 and Figure 1-5. Process definition “Loan” was simulated under two Application Databases with the different Replica IDs.

For each process definition that has been simulated in a certain Application Database, system performance data associated with the process definition are

---

#### Example Table

<table>
<thead>
<tr>
<th>No</th>
<th>Process</th>
<th>Application Database</th>
<th>Life Period (Jobs)</th>
<th>Total Created Jobs</th>
<th>Finished / Stopped Jobs</th>
<th>Running Jobs (Average)</th>
<th>Duration (Average)</th>
<th>Iteration</th>
<th>Human Costs</th>
<th>Material Cost</th>
<th>Fixed Costs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Order</td>
<td></td>
<td></td>
<td>317</td>
<td>258</td>
<td>98.85</td>
<td>38988.18</td>
<td></td>
<td>£2884.17</td>
<td>£3.00</td>
<td>£104.00</td>
</tr>
<tr>
<td>2</td>
<td>Report</td>
<td></td>
<td></td>
<td>317</td>
<td>127</td>
<td>242.05</td>
<td>15450.53</td>
<td>100 Times exceed 1</td>
<td>£13943.92</td>
<td>£3.00</td>
<td>£1043.00</td>
</tr>
<tr>
<td>3</td>
<td>Loan [Design 2]</td>
<td>C12567890000123456789</td>
<td>01/01/2000 00:00</td>
<td>50</td>
<td>35</td>
<td>22.14</td>
<td>48985.07</td>
<td></td>
<td>£3723.09</td>
<td>£3.21</td>
<td>£224.00</td>
</tr>
<tr>
<td>4</td>
<td>Loan</td>
<td></td>
<td></td>
<td>316</td>
<td>210</td>
<td>128.95</td>
<td>58375.97</td>
<td></td>
<td>£22271.91</td>
<td>£187.74</td>
<td>£1104.00</td>
</tr>
</tbody>
</table>
summarized in the table from the activity report and the resource report. Statistical data for a simulated process definition includes:

- the Application Database in which the process definition was simulated,
- life period specification of the process definition,
- total simulated process instances,
- number of terminated and stopped process instances,
- average number of running process instances in the system,
- average duration of a process instance,
- total costs of human and material resources,
- total fixed costs, etc.

The report includes also escalation data in order to raise necessary alarms for the simulated WfMS. Escalation values relevant to a process definition are accumulated when a process instance is terminated or stopped. In this example, of all 127 completed process instances in accordance with process definition “Report”, no process instance has multiple execution iterations of any activity within the process instance. Some process instances with multiple iterations may have not completed when the simulation run ended after 12/31/2001.

### 13.3.3 Activity Report

The activity report as shown in Figure 13-15 collects statistical data of all activities within simulated process definitions. For each activity there are:

- fixed costs,
- total number of created and completed activity instances,
- average number of running instances of the activity in the system,
- average number of team members having executed the activity;
- average execution time and delay time for an instance of the activity by a workflow participant,
- average join-waiting time,
- average material availability-waiting time,
- average material release-waiting time,
- average duration of an instance of the activity,
- number of process instances containing execution iterations of the activity exceeding the maximum number of iterations (here 1) specified to the process definition, etc.

Fixed costs are directly proportional to the number of executed activity instances. If execution iteration of an activity happens, total number of the activity instances can be greater than the simulated process instances associated
with the same process definition, since a process instance experiences repetitive executions of the activity.

Waiting for joining happens only at a join activity. Waiting for material availability occurs if a start-synchronous material is not available for a workflow participant to execute an activity. Waiting for material release appears when a workflow participant has completed an activity but can not route the work associated with the activity further, because a routing-synchronous material has not been released from execution of that activity.

Average duration of an activity is affected by the execution time, delay time, join-waiting time, material availability-waiting time, and material release-waiting time. From this report, the reasons of long duration of an activity can be analyzed.

1. Long execution time and delay time are required. Note that, if the average number of team members executing the activity is larger than one, the sum of average execution time and delay time may be larger than average duration, because multiple workflow participants of a team can execute the activity simultaneously.

Figure 13-15. Activity Report
2. **A lot of time waiting** for joining. For activity “Approve credit” in the Application Database with Replica ID “C1256768:00377BA7” (item 12 in the table), average execution time is 85.71 minutes, join-waiting time is 12334.70 minutes and duration 39971.76 minutes. That is, the long duration is caused in a great part by waiting for joining. The process definition can be improved by balancing parallel execution threads;

3. Much time waiting for material availability. There are not enough materials available to be used for execution of the activities.

4. Long time waiting for material release. Execution time and material occupation time are not balanced, or there is a shortage of material resources. For activity “Inform applicant” (item 23), averagely 7.10 minutes waiting for material release contributing to the average duration of 23.58 minutes;

5. Shortage of workflow participants for executing the activity, if none of above 1 to 4 factors cause a long duration, such as activity “Register order”. From the resource report (see Section 13.3.4), the workflow participants who executed the activity can further be analyzed.

Escalation statistic variables relevant to an activity are accumulated. If there are a lot of iterations of the activity execution, duration of a process instance will be prolonged and fixed costs and resource costs will be increased.

### 13.3.4 Resource Report

The resource report as shown in Figure 13-16 indicates performance data of each simulated human and material resource, retrieved from the configured Organization Database and/or Notes Organization Directory configured to the Application Database in which a process definition is simulated.

Statistic data for each simulated resource includes:

- utilization rate during the simulation period,
- average queue length of the resource during the simulation period,
- average waiting time of a work in the queue before it is executed,
- total number of completed work items associated with different activities,
- total execution time, and
- costs for these executions.

Resource cost is proportioned to total executed time since hourly costs of a resource can be retrieved from the configured Organization Database.

If a resource has been demanded by too many work items associated with various activities, its queue length and waiting time will rise, and performance of the system becomes worse. However a low utilization rate of a resource means waste of resource costs. Usually a resource with long waiting queue has
high utilization rate. If not, balance problems may exist between human and material resource, both are required simultaneously for activity execution.

The table can be sorted in ascending or descending order for each column. Data in the table can be presented in chart form as shown in Figure 13-17 for

**Figure 13-16. Resource Report**

**Figure 13-17. Resource Report—Chart**
easy comprehension and comparison.

The resource report is the most important report to help the system analyst to detect potential resource bottlenecks in the simulated WfMS. The bottlenecks may be caused by the shortage of resources at the top of the table after it is sorted in descending order on the columns "Utilization Rate", "Average Queue length" or "Average Waiting Time".

From the example, we see in Figure 13-17 that Suelyn Hayes is one of the bottlenecks of the system. She has nearly 100% utilization rate, and much longer waiting queue than that of any of the other simulated people. The work items in her worklist wait much more time before they can be executed. To reduce her work associated with the simulated process definitions can improve the system performance.

### 13.3.5 Work Allocation Report

The work allocation report as shown in Figure 13-18 presents the distribution...
of the work items associated with the simulated process definitions among human or material resources. Upon this report, it can be analyzed how work items associated with each activity are executed by members of a team editor assigned to the activity, and how a resource has undertaken different activities. The data in the report can be displayed in chart form (see Figure 13-19 and Figure 13-20) oriented to an activity or to a human/material resource for ease of comparison and analysis.

In this example, Suellyn Hayes has participated in the work associated with all the three simulated process definitions for executing activities of “Check order”, “Notification”, “Work on report”, “Evaluate”, “Approve credit” and “Check personal creditworthiness”. This can be seen more clearly in Figure 13-19. Obviously, Suellyn Hayes has done more work for activity “Work on

![Figure 13-19. Work Allocation Report—Resource-Oriented](image)

![Figure 13-20. Work Allocation Report—Activity-Oriented](image)
report” than any other activities. To prevent her from executing the activity can reduce her work.

Figure 13-20 presents graphically how activity “Work on report” has been executed by all team members of department “Cascadia”—Suelynn Hayes has executed all instances of the activity assigned to the team. If a workflow participant executes an activity assigned to a team editor much more than other team members, to reduce his work for this activity, we should first know the reason why this happens. Should he complete the activity? Has he higher probability to execute the activity?

If all members resolved from an editor are potential bottlenecks of a WfMS, the editor is then a potential bottleneck. That is, the editor has been assigned to too much work associated with the activities within the simulated process definitions. The system performance might be improved through:

1. assigning some activities to other editors,
2. separating an activity with long executing time into two or more small activities and assigning them to different editors, or
3. increasing members of the team editor in the organization and making most of the members have opportunity to execute work associated with different activities, provided that the activities are not completed by all team members.

13.4 Events in Espresso Simulation Model

The next-event approach is implemented in the Espresso PM as the model algorithm for simulating multiple process definitions in different Application Databases dynamically, graphically, simultaneously and interactively. The eventlist (the sequence of events with respect to occurrence time) is of particular importance for keeping information about expected changes of the system states.

13.4.1 Scheduled Events

According to the first occurrence of the scheduled events kept in the eventlist, the simulation model with the next-event approach advances the clock and proceeds a simulation run. The scheduled events in the Espresso simulation model are the arrival event, the participant depart event, the material release event, the routing step event, the delay finish event and the simulation termination event.
• **Arrival event**: a process instance is created at a start activity of a process definition, or a routed work associated with a process instance arrives at an activity. It is scheduled

1) at the beginning of a simulation run (one or the specified number of process creation events are scheduled for each start activity);
2) when an arrival event happens at a start activity (that is, when a process creation event happens) and it is the last scheduled at the beginning of the simulation run or is scheduled during the simulation run;
3) when a routing step event happens and the next step along a link will arrive at the destination activity of the link;
4) when calling the algorithm for Routing Work to Activity and the work can be directly routed to a successor activity.

The schedule of arrival events at a start activity (see Assumption 11-1) is constrained by the life period settings of the process definition and the simulation beginning/ending conditions.

• **Participant depart event**: a workflow participant completes an activity. It is scheduled when the workflow participant begins to execute an activity. The execution time is generated when the work associated with the activity enters the waiting queue of the workflow participant.

• **Material release event**: a material finishes the occupation for execution of an activity. It is scheduled when a material begins to be used for execution of an activity. The occupation time is generated when the demand for using the material is enter the waiting queue of the material.

• **Routing step event**: to animate a step of a routed work along a link. It is scheduled

1) when a work can be routed to another activity and the given number of steps is larger than one;
2) when a routing step event happens and the next step does not arrive at the destination activity.

• **Delay finish event**: the required delay time of an activity is just elapsed. It is scheduled when a workflow participant prepares to execute an activity but a specified delay is required before executing the activity. The delay time is generated when the work associated with the activity enters the waiting queue of the workflow participant.

• **Simulation termination event**: to terminate a simulation run and to generate the simulation reports. It is scheduled when a specified end activity is completed during animation. It can occur

1) when the clock exceeds the given simulation period and not all running process instances should be finished, or
2) when the eventlist is empty as the simulation model advances the clock (at this time, the given simulation beginning/ending conditions have been met and all running process instances have been finished), or
3) when the simulation run is interrupted by the system analyst.

13.4.2 Conditional Events

Some events or state changes in the simulation model are conditional on the occurrence of other events. So they do not need to be scheduled.

A work associated with an activity can be routed to a successor activity (that is, the algorithm for Routing Work to Activity is called) if all workflow participants of a team finish execution of the same activity instance and all routing-synchronous materials are released from the execution of the activity instance. This condition is met when a participant depart event or a material release event happens.

A workflow participant can begin executing an activity (that is, a participant occupation event occurs), if he is a dummy participant or a Notes agent, or he is idle and there is a work in his waiting queue and all the start-synchronous materials required for the activity execution are available. This condition will be checked

- when a work is allocated to the waiting queue of a workflow participant;
- when a participant depart event happens;
- when a delay finish event happens; or
- when a material release event happens and a workflow participant is waiting for availability of the material.

If an activity prepared to be executed requires a delay and the delay time has not elapsed, the activity can not be executed at once and a delay finish event will be scheduled for the activity.

A work is allocated to the waiting queue of a workflow participant (that is, a work allocation event occurs)

- when an arrival event happens and it is determined that the work arriving at the activity does not need waiting for joining; or
- when a deadlock is released during detecting deadlock (that is, when calling the algorithm for Releasing a Deadlock).

The algorithm for Detecting Deadlocks will be called,
• when an arrival event happens and it is determined that the work arrived at the activity must wait for joining;
• when a parallel work can be routed to another activity (that is, the algorithm for Routing Work to Activity is called); or
• when a parallel work is completed at a structural end activity of the process definition (that is, a process instance has parallel execution threads and one of them is terminated or stopped).

A material resource begins to be used (that is, a material occupation event occurs), if the waiting queue of the material is not empty. This situation is checked,

• when a material release event happens; or
• when a participant occupation event is processing.

Figure 13-21 summarizes the relationships between scheduled and conditional events. The participant occupation event, the work allocation event,
and the material occupation event are dependent on other events or algorithm modules. Thus they do not need to be scheduled.

13.5 Conclusion

Three alternative graphical simulation modes can be run in PM. Animation of one process definition let the system analyst interactively simulate a process instance from a specified start activity till the end activities of the process definition or a specified activity. Simulation of one or all opened process definitions simultaneously in different Application Databases let a WfMS be simulated as it is constructed in the real world.

Various simulation settings are required for a simulation run. Some are defined with activities, links and resource databases; some are specified particularly for the simulation study to a process definition, an Organization Database or a Notes Organization Directory. Before a simulation run, the system analyst determines in the experiment settings the unit of the clock, simulation run length, protocol-generating intervals, and the Application Databases as well as configured Organization Databases and/or Notes Organization Directories for each simulated process definition. The experiment settings can be saved as a scenario for later simulation runs. The view settings for graphical display are specified before a simulation run and can be modified during simulation.

Dynamic process protocols can be generated during simulation and the statistical reports of the summary report, the activity report, the resource report and the work allocation report can be generated after a simulation run. The process protocol presents in detail an execution thread of a process instance from the start activity till the specified or end activity. The summary report presents general results of a simulation run such as the simulated process instances, simulation period and total costs. The activity report collects the data of each simulated activity, such as duration, costs, various waiting times, etc. The resource report indicates performance (such as utilization rate, queue length, waiting time, costs, etc.) of each simulated human and material resource. The work allocation report displays the distribution of work items associated to various activities among simulated human or material resources. The simulation reports help the system analyst to study the performance of a WfMS and to detect bottlenecks in the system, in order to improve the system efficiency via modifying the process definitions.

The events occurring in a WfMS make the system states change. The next-event approach is implemented in the Espresso simulation model as the main simulation procedure of a simulation run. The scheduled events in the simulation model are the arrival event, the participant depart event, the material release event, the routing step event, the delay finish event and the simulation termination event. A simulation run terminates when there is no scheduled event in the eventlist as the simulation model advances the clock.
13.6 Further Work*

The following points are not implemented in the Espresso simulation model. For a more practical and flexible simulation tool, they could be considered for further work.

- **Warm-up period**
  To compare simulation results of dynamic systems with theoretical results (see Section 14.3), it is desired to segment a simulation run into an initial portion, called *warm-up period*, and a later portion on the supposition that some time will elapse before the model is in a steady state condition. The warm-up period may exhibit transient conditions such as an abnormally low average time in queue because the run begins with no process instance in the system and should be dissipated before beginning to collect statistical information on system performance, so that the statistical variables are accumulated during the simulation based upon steady-state behavior.

  “There is no particular criterion for establishing the length of the warm-up period, so that its length is more or less arbitrary. Typically a warm-up period may be anywhere form 5 percent to 20 percent of the remaining (steady-state) run.” [Gottfried, 1984, p. 178].

  Warm-up period can also be used for simulating an existing system, whose historical running data should not be included in the calculation of system performance.

  The warm-up period might be included in the life period specification for each process definition.

- **Multiple values of a variable within a distribution interval**
  In the current simulation model, the assignment of multi-values to a variable defined in a “Condition” link allows only one value from each subinterval of the given variate distribution function followed by the variable. In real world business processes however, it does not always work so. For example, in Figure 11-4, the assignment of multi-values “82900:1693:59277886, 4000” to a Notes field is feasible in a WfMS, where both 1693 and 4000 are in the same interval (1000,10000). This constraint should be eliminated for the further work.

- **Regional public holidays and personal holidays**
  Regional public holidays (that is, the holidays that are not the common holidays of all simulated Organization Databases and Notes Organization Directories) and personal holidays should be considered in the simulation model and can be treated in the same way. If a person begins executing an activity, the time taken for the execution of the activity will be prolonged if his personal holidays or regional public holidays occur before he completes the activity.
• Simulating stand-in
If personal holidays could be considered in the simulation model, stand-in might also be accounted for in the simulation study. The rules for a stand-in to take over a work must then be given.

• Simulating capacity of a Notes agent
A Notes agent as an IT workflow participant running on a client or a server has in fact capacity limit and should be considered in the simulation model.

• Team members execute an activity synchronously (Team work)
In the real world WfMS, it can happen that an activity (a meeting for example) must be executed by all the workflow participants of a team at the same time. If this could be simulated, whether the specified material resources should be used for each individual workflow participant or for the whole team can also be simulated.

• Empirical distribution for input of the simulation model
If the distribution function is possible to be specified to a single time variable, for example, the execution time of a certain activity within a process definition, the time variable can then be specified to follow an empirical distribution.

• Queuing rules
“The queue is a set of jobs that are waiting for service. We may think of the queue as a list from which arrivals are selected for service according to a rule called the queue discipline. For example, jobs may be selected as follows:

1. in the order of their arrivals (first-come-first-served),
2. in the reverse order of their arrivals (last-in-first-served),
3. in the order of shortest service time,
4. in the order of longest service time, or
5. according to a priority number that each job has for service.”
   [Fishman, 1973, p. 31]

For the further work, it should also be possible to queue a worklist in the simulation model according to the due date or the priority of a process instance, since due date and a priority can be specified to a process instance in the Espresso WfMS.

Furthermore, if there are multiple process definitions enacted in a WfMS, the business processes associated with some process definitions are more important than those of other process definitions. So the simulation model should be able to specify priority to a process definition, and a workflow
participant can then execute the work in the order of the priority of the associated process definition.

Although the worklist in the Espresso WfMS is sorted in a certain rule, some workflow participants may do the work in a random order. This situation should also be possible to simulate. The queuing rule can then be different for different workflow participants.

- **Multiple waiting queues for a resource**
In a real world WfMS, the worklist may be categorized into several worklists (waiting queue). The polling discipline for a resource to select a queue could be specified according to (see [Mittra, 1986, p. 187]):

1. the order in which the queues are selected;
2. the number of work items served at each polling session; and
3. the time in transferring service among the queues.

- **Change input of the simulation model during a simulation run**
The input variables of the simulation model must be determined before a simulation run and cannot be changed during the run. But in a real world system, especially for a system with a long life period, some system parameters and decision variables may change at a certain point in time. So the simulation model should be able to simulate this case.

- **Keep input of a simulation run in the scenario**
In addition to the experimental settings can be saved as a scenario, other input variable of a simulation run, such as process definitions, process settings and organizational settings, should also be included in the scenario, so that the saved simulation reports are always corresponding to the input data kept in the scenario.

- **Graphical display of state variables**
One or multiple state variables of the simulation model can be graphically presented, such as the plot of a queue length shown in Figure 8-3. These variables can be specified with the experimental settings of a simulation run.
14 GENERAL SIMULATION PHASES

To simulate and analyze the system performance of business processes in accordance with the process definitions, the following general simulation phases will be encountered by a system analyst:

- collect history data;
- determine input data of the simulation model;
- validate of the simulation model;
- evaluate simulation results;
- simulate alternative operating policies and select the best one for implementation of a WfMS.

Before a simulation run, history data of a simulated system should be collected for validation of the simulation model and estimation of the input variables to the simulation model.

Once the simulation model is validated, alternative operating policies can be designed (see [Naylor, 1969, pp. 3-120]) for simulating one after another. The simulation results can then be assessed based upon the science of statistics. The best operating policy among the simulations can be decided via analyzing the statistically significant difference between the means of the performance index of the system for any two alternatives.

14.1 History Data Collection

“Probably the least glamorous and most essential task in model building is gathering the data which will permit the analyst to estimate model parameter.” [Solomon, 1983, p. 11].

To run the simulation model of a WfMS, much data are required for specifying various system parameters and decision variables. Some data are also gathered for validating a simulation model. For example, historical distribution of process creation times can be used to estimate the process intercreation distribution; collected activity execution time and delay time can be used directly for the definition of an activity; observed activity duration, waiting time, queue length and resource idle time can be used to validate the simulation model.

Recording data of an existing system for simulation input data and model validation could be burdensome and prone to error. To minimize difficulties, at least two people should share the data-keeping duties—one to observe and describe the system activities, the other to operate a stopwatch and record system performance data. It is helpful if a form for data collection is prepared in advance of observing the system and if the base time unit (e.g., minutes,
If a process definition consists of multiple activities, accuracy might be best achieved by breaking the system down into activities that can be observed separately, by other teams or on other occasions. If data collection is to take place on different occasions, the analyst should endeavor to make sure that the occasions are comparable. For example, if it is desired to model system performance at a peak traffic hour, additional data which may be needed for the model should not be collected at a normal or slow time, unless these conditions are to be modeled separately (see [Solomon, 1983, p. 13]).

The analyzer integrated in PM as shown in Figure 14-1 can help the system analyst to record data of running (as well as completed) process instances in accordance with a process definition. The most useful data can be gathered by the analyzer are:

- duration of all completed process instances;
- creation time of all process instances;
- the running activity instances associated with a process definition;
- worklist of each workflow participant; etc.

The data collected by the analyzer can also be graphically displayed as shown in Figure 14-2. The data gathered regularly from an existing WfMS can be used to determine some input variables of the WfMS system that will be simulated in the future or to validate the simulation model of the PM.
14.2 Distribution Function Selection

To formulate the Espresso simulation model as a representation of the real world WfMS, it is necessary for the system analyst to specify various assumed distributions for generating process intercreation time, activity execution/delay time, work routing time, material occupation time, and other random variates. “To be useful, the assumed form should be sufficiently realistic, so that the model provides reasonable predictions while, at the same time, being sufficiently simple, so that the model is mathematically tractable.” [Hillier, 1974]

If the behavior of an element can not be predicted exactly, given the state of the system, it is better to take random observations from the probability distributions involved than to use averages to simulate this performance. This is true even when one is only interested in the average aggregate performance of the system because combining average performances for individual elements may result in something far from average for the overall system. See [Hillier, 1974, p. 625]

One question that may arise when choosing probability distributions for the model is whether to use frequency distributions of historical data or to seek the theoretical probability distribution which best fits these data. The later
alternative usually is preferable because it would seem to come closer to predicting expected future performance rather than reproducing the idiosyncrasies of a certain period of the past. See [Hillier, 1974, p. 625].

The choice of a distribution function is sometimes troublesome to the beginning practitioner. Therefore some guidelines should be helpful.

[Gottfried, 1984, p. 102] indicated four considerations in the choice of a distribution function for a random variable:

1. special characteristics of a particular distribution function;
2. accuracy with which a distribution function can represent a given set of empirical data;
3. ease with which a distribution function can be fitted to a given set of empirical data;
4. computational efficiency when generating random variates.

The exponential distribution is frequently chosen to represent random arrivals to a system because of its special applicability to such situations. Moreover exponentially distributed random variates can be generated efficiently. The use of this distribution function is therefore recommended without reservations for those situations to which it applies. The normal distribution is also used extensively because so many naturally occurring phenomena seem to be governed by this distribution. Unfortunately it is less efficient to work with than the exponential. The gamma distribution is often used to represent a skewed set of empirical data, this is not a convenient function to work with, however, since it cannot easily be fitted to empirical data (nonlinear regression is required), and its use is relative inefficient from computational standpoint. In many practical applications, however, an empirical distribution will be entirely adequate; such distributions can easily be fitted to empirical data, and are computationally efficient. See [Gottfried, 1984, pp. 102-103].

The $\chi^2$ test give us a method for determining the appropriateness of assuming that a random variable is governed by a certain distribution function. See Section 9.5 for an example about how to fit a distribution to a given set of empirical data.


14.3 Validation of the Simulation Model*

“By validation we mean a study of how well the behavior of a model accords with that of the true system.” [Fishman, 1973, p. 311]

The simulation model consists of a high number of variables, assumptions, and cause-and-effect relationships. Therefore, even when the individual components have been carefully tested, numerous small approximations can still accumulate into gross distortions in the output of the overall model. Consequently, it is important to test the validity of the model for reasonably predicting the aggregate behavior of the system being simulated. Only an accurate model that contains an adequate level of detail can encourage a decision-maker to use the model for analyzing a system and for making decisions upon the simulation results.

“In order to determine the validity of a simulation model we must first recognize the following likely sources of error:

1. The data. This refers to both the accuracy of the data and the type of data (that is, the particular parameters that were measured).
2. The model itself. Here we are primarily concerned with the assumptions that were used to build the model, and the validity of the cause-and-effect relationships among the variables.
3. Implementation of the model. This is largely a matter of programming accuracy.
4. Interpretation of the results.

Each of these items should be examined carefully if the accuracy of the model is considered to be in question. Each item is fundamentally distinct, however, which precludes the use of simple standardized procedures for error detection. Thus, the analyst must examine each item carefully and critically, applying sound judgment, common sense, and attention to detail. A good deal of time and patience may be required for this phase of the work.” [Gottfried, 1984, p. 179]

See [Lehman, 1977] for more discussions about the validation.

14.3.1 Determination of Expected System Performances

Perhaps the easiest way to assess the validity of a simulation model is to simulate the expected behavior of a system whose performance characteristics are known. Comparisons can then be made between the simulated and the expected data. See [Gottfried, 1984, p. 180].

Standard statistical tests can sometimes be used to determine whether the differences in the means, variances, and probability distributions generating the two sets of data are statistically significant. The time-dependent behavior of the
data might also be compared statistically. If the performance characteristics are not amenable to statistical analysis, personnel familiar with the behavior of the real system should be asked if they can discriminate between the two sets of data. See [Hillier, 1974, p. 633].

There are several ways in which the expected performance characteristics of a system can be determined (see [Gottfried, 1984, p. 180]).

- Theoretical predictions: used only for simple, idealized models, such as idealized queuing systems. With the Espresso PM, the system analyst can design a corresponding process definition. To compare the simulation results of a process definition with the theoretical predictions can validate the simulation model.

- Hand calculations: tends to be limited to simple systems with a small number of random events, since it is generally impractical to carry out an extensive amount of computation manually. On the other hand, some occasional hand calculations can provide considerable insight into the details of the computational procedures as well as establish a basis of comparison for the computerized simulation model. Such calculations can be very useful, particularly for debugging purposes.

  Hand calculations for the results of simulating a pair of process instances in accordance with a process definition with fixed distribution function specification are not complicated. So the Espresso simulation model can be partly validated via hand calculations.

- Historical data: generally refer to information that has been obtained for an existing system whose behavior is well understood. The assessments of a simulation model based upon such historical data tend to be much more subjective than the other two methods. Nevertheless, historical assessments are frequently of greater value because they can be applied to more realistic and complex types of situations. For many problems the successful simulation of known past performance is a critical test that usually enhances one’s confidence in the validity of the simulation model.

Without any real data as standard of comparison, the only way to validate the overall model is to have knowledgeable people carefully check the credibility of output data for a variety of situations. Even when no basis exists for checking the reasonableness of the data for a single situation, some conclusions usually can be drawn about how the relative performance of the system should change as various parameters are changed.

It is especially important to convince the decision-maker of the credibility of the simulation model, so he will be willing to use it to aid his decisions. If the model may be used again in the future, keeping the actual results of an implemented WfMS is very important for model validation and input data determination in the future.
14.3.2 Some Theoretical Results for the Single-channel Single-station Queue

The single-channel single-station queue has been studied theoretically for certain conditions that are of practical interest. Some of the results that are obtained are subsequently summarized. The theoretical results for the single-channel single-station queue can be used to validate the simulation model in PM by comparing with the simulation results.

The behavior of a WfMS resembles a queuing or waiting line problems. In a system of queuing problem, an arrival occurs and demands that a service be performed. The system responds by performing the service if it can, or by keeping the demand waiting until it can perform it. The simplest queuing problem is a system with the single-channel, single-station queue as shown in Figure 14-3. There is one station performing the service and one queue waiting for the service in the system.

Two groups of theoretical results of the queue system will be discussed. For more discussions about various queue models see [Tijms, 1986] and [Kleinrock, 1996].

14.3.2.1 The M/M/1 Queue

Consider a single-channel, single-station queue where the interarrival times are exponentially distributed with mean $\lambda$ and the service times are exponentially distributed with mean $\mu$. Let

$$\beta = \mu / \lambda$$

The following results can then be obtained provided $\beta < 1$ (see [Gottfried, 1984, p. 204]):
• Expected waiting time = $\mu\beta / (1 - \beta)$
• Expected time spent in the system (waiting time + service time) $= \mu / (1 - \beta)$
• Expected queue length = $\beta^2 / (1 - \beta)$
• Expected nonempty queue length = $1 / (1 - \beta)$
• Expected fraction of time the server is idle = $(1 - \beta)$

If $\beta > 1$, the system will be unstable (arrival occurs frequently than a service is completed). Under these conditions the queue will continue to grow with time. Moreover, if $\beta = 1$, the queue length will oscillate with time. These undesirable situations should be avoided if at all possible.

14.3.2.2 The M/G/1 Queue

Now consider the case where the interarrival times are exponentially distributed with mean $\lambda$ as before, but the service times are governed by a two-parameter distribution having mean $\mu$ and standard deviation $\sigma$, for example, a normal distribution. We again define

$$\beta = \mu / \lambda$$

The following results can be obtained provided $\beta < 1$ (see [Gottfried, 1984, p. 204]):

• Expected waiting time $= (\sigma^2 / \lambda + \beta^2\lambda) / (2(1 - \beta))$
• Expected time spent in the system (waiting time + service time) $= \mu + (\sigma^2 / \lambda + \beta^2\lambda) / (2(1 - \beta))$
• Expected queue length $= (\sigma^2 / \lambda^2 + \beta^2) / (2(1 - \beta))$
• Expected fraction of time the server is idle $= (1 - \beta)$

Example 12-1
Consider a system has one consulting station that has exponential interarrivals with a mean of 30 minutes, and normal service times with a mean of 20 minutes and a standard deviation of 5 minutes. That is,

$$\lambda = 30, \mu = 20, \text{ and } \sigma = 5$$

So

$$\beta = \mu / \lambda = 20 / 30 = 0.667$$
The theoretically expected values of the system can then be determined as:

\[
\begin{align*}
\text{Expected waiting time} &= \left( \frac{\sigma^2 / \lambda + \beta^2 \lambda}{(2(1 - \beta))} \right) \\
&= \left( \frac{(5)^2 / (30) + (0.667)^2(30)}{(2(1 - 0.667))} \right) \\
&= 21.3 \text{ (minutes)} \\
\text{Expected queue length} &= \left( \frac{\sigma^2 / \lambda^2 + \beta^2}{(2(1 - \beta))} \right) \\
&= \left( \frac{(5)^2 / (30)^2 + (0.667)^2}{(2(1 - 0.667))} \right) \\
&= 0.71 \text{ (arrivals)} \\
\text{Expected fraction of time the server is idle} &= (1 - \beta) \\
&= (1 - 0.667) \\
&= 0.333
\end{align*}
\]

Now we model a process definition with a single activity and specify one person to execute the activity with execution time 20 minutes and deviation 5. Let the intercreation time between two consecutive process creations meet the exponential distribution with mean 30 minutes and the organization to which the person belongs has normal distribution for activity executions. We simulate 5000 process instances three times and get the results as shown in Figure 14-4.

<table>
<thead>
<tr>
<th>Simulation - Resource Report</th>
<th>Person</th>
<th>Material resource</th>
<th>PopUp</th>
<th>Descending</th>
<th>Chart</th>
<th>Print...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Utilization</td>
<td>Queue</td>
<td>Waiting Time</td>
<td>Finished</td>
<td>Processing</td>
<td></td>
</tr>
<tr>
<td>Hong Zhang</td>
<td>60.35</td>
<td>0.45</td>
<td>15.11</td>
<td>5000</td>
<td>100230.00</td>
<td></td>
</tr>
<tr>
<td>Hong Zhang</td>
<td>66.01</td>
<td>0.52</td>
<td>18.94</td>
<td>5000</td>
<td>100538.00</td>
<td></td>
</tr>
<tr>
<td>Hong Zhang</td>
<td>73.30</td>
<td>0.98</td>
<td>26.90</td>
<td>5000</td>
<td>99713.00</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 14-4.** Simulation Results of a M/G/1 Queue

Summarize the results to the following table for comparing with the theoretical results:

<table>
<thead>
<tr>
<th></th>
<th>Theoretical prediction</th>
<th>Run 1 Results</th>
<th>Run 2 Results</th>
<th>Run 3 Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of simulated process instances</td>
<td>((\infty))</td>
<td>5000</td>
<td>5000</td>
<td>5000</td>
</tr>
<tr>
<td>Mean waiting time (minutes)</td>
<td>21.3</td>
<td>15.11</td>
<td>18.94</td>
<td>26.90</td>
</tr>
<tr>
<td>Mean queue length</td>
<td>0.71</td>
<td>0.45</td>
<td>0.62</td>
<td>0.98</td>
</tr>
<tr>
<td>Fraction of time the server is idle</td>
<td>0.333</td>
<td>0.3965</td>
<td>0.3399</td>
<td>0.2670</td>
</tr>
</tbody>
</table>

The agreement appears reasonable, thus enhancing our confidence in the simulated results and the simulation model.
14.4 Evaluation of Simulation Results*

“Simulation should not be regarded as a panacea. A simulation model includes uncertain events. Hence the answers it provides should be regarded as approximations subject to statistical error.” [Mitra, 1986, p. 172] If a random variable is included in the input data of simulation model, a related system performance criterion value obtained from a simulation run, such as average duration of a process instance, is also random and can not be considered as the unique value of the criterion. We can, however, determine the interval within which the mean value of the system performance criterion falls, at a given confidence level. To do so, we first define the following symbols:

\[
\bar{Y} = \text{the calculated mean value (that is, the sample mean) of the system performance criterion;}
\]

\[
s = \text{the calculated standard deviation of the system performance criterion;}
\]

\[
n = \text{the number of simulated values of the performance criterion used to calculate } \bar{Y} \text{ and } s;
\]

\[
\mu = \text{the true mean value of the system performance criterion (which is unknown).}
\]

Here

\[
\bar{Y} = \frac{1}{n} \sum_{i=1}^{n} Y_i
\]

and

\[
s^2 = \frac{1}{n} \left( \sum_{i=1}^{n} Y_i^2 \right) - (\bar{Y})^2
\]

If the calculated \( Y_i, i = 1, 2, ..., n \), is normally distributed, then the statistic

\[
\frac{\bar{Y} - \mu}{s} \sqrt{n-1}
\]

is known to follow \( t \)-distribution with \( n - 1 \) degrees. If \( Y_1, Y_2, ..., Y_n \) are independent and have a symmetric probability distribution function, the statistic can be approximated to the \( t \)-distribution. See [Fishman, 1973, pp. 263-268]. So

\[
\bar{Y} - t_{n-1,1-\alpha/2} s / \sqrt{n-1} < \mu < \bar{Y} + t_{n-1,1-\alpha/2} s / \sqrt{n-1}
\]
Here \((1 - \alpha)\) is the corresponding confidence level.

**Example 12-2**

The process definition in the Figure 1-3 was simulated for 100 process instances with the specification that link “Order Accepted” has 100% routing probability (and so another “Exclusive Choice” link “Order Denied” has no possibility to let a work route along). The duration \(Y\) of a process instance from the start activity to the completion of the end activity will be evaluated. After the simulation run, the average duration \(\bar{Y}\) was 1168.51 minutes with \(s = 60.38\).

Determine the range of process duration \(\mu\) corresponding to a 95% confidence level, assuming that the individual \(Y\)'s are normally distributed.

In this example we know that \(\alpha = 0.05\) and \(n = 100\). Hence we can obtain an appropriate value for statistic \(t\) from Table 1, using linear interpolation between the tabulated values for

\[
t_{60, 0.975} = 2.00
\]

and

\[
t_{120, 0.975} = 1.98
\]

Thus

\[
t_{99, 0.975} = 2.00 + \left(\frac{99 - 60}{120 - 60}\right)(1.98 - 2.00) = 1.987
\]

The range of \(\mu\) can now be determined as follows:

\[
\bar{Y} \pm t_{99, 0.975} \cdot s / \sqrt{n - 1} = 1168.51 \pm (1.987)(60.38) / \sqrt{99}
\]

\[
= 1168.51 \pm 12.06
\]

Therefore

\[
1156.45 < \mu < 1180.57
\]

We conclude that the true mean value for duration of a process instance \(Y\) falls between 1156.45 minutes and 1180.57 minutes at 95% confidence level.
14.4.1 Determination of Sample Size

In most realistic situation studies, we wish to determine a value of $n$ that will allow the true mean $\mu$ to fall within a desired interval at a specified confidence level. This can be accomplished by assuming that the calculated mean $\overline{Y}$ and standard deviation $s$ will not change appreciably as $n$ is increased. Thus $n$ can be solved directly once $\overline{Y}$ and $s$ have been determined. See [Gottfried, 1984, p. 171]. From the central limit theorem, the true mean can be estimated in the range

$$\overline{Y} \pm Z_{0.5-\alpha/2} \frac{s}{\sqrt{n}}$$

when $n \geq 30$. So if the desired interval is expressed as $\overline{Y} \pm \theta$, then

$$Z_{0.5-\alpha/2} \frac{s}{\sqrt{n-1}} = \theta$$

Solving for $n$, we obtain

$$n = (Z_{0.5-\alpha/2} s/\theta)^2 + 1$$

Thus given the interval $\theta$, the least sample size $n (>30)$ can be determined, so that the mean $\mu$ falls within the range $\overline{Y} \pm \theta$ at a specified confidence level $100(1-\alpha)\%$.

Example 12-3

We simulate the process definition described in Example 12-2 with different values of sample size $n (=10, 100, \text{and} 1000)$ and for each given value $n$ two times are simulated. The simulation results after the six runs are summarized in the following table:

<table>
<thead>
<tr>
<th></th>
<th>$n = 10$</th>
<th>$n = 100$</th>
<th>$n = 1000$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\overline{Y}$</td>
<td>$s$</td>
<td>$\overline{Y}$</td>
</tr>
<tr>
<td>Run 1</td>
<td>1147.90</td>
<td>58.70</td>
<td>1161.59</td>
</tr>
<tr>
<td>Run 2</td>
<td>1145.10</td>
<td>58.19</td>
<td>1170.17</td>
</tr>
</tbody>
</table>

Thus it can be concluded that the calculated mean and standard deviation will not change appreciably as $n$ is increased, especially when $n > 100$.

Example 12-4

Again consider the duration problem described in Example 12-2. We wish to simulated a large enough number $n$ of process instances so that the true mean value $\mu$ of duration $Y$ falls within $\pm0.5\%$ of the calculated mean $\overline{Y}$, at a 95% confidence level.
From Example 12-2, we have already seen that $n$ must exceed 100, because we obtained an interval size of $1168.51 \pm 12.06$ when $n = 100$, and the interval $\pm 12.06$ corresponds to a value of $\pm 1.03\%$ (larger than the desired $\pm 0.5\%$) of the calculated mean $\bar{Y} (= 1168.51)$.

Let us assume that the calculated mean will remain approximately equal to 1160, and the standard deviation 60. The specified interval bound can then be expressed as

$$\theta = 0.005 \bar{Y} = (0.005)(1160) = 5.8$$

Hence

$$n = (Z_{0.5 - \alpha / 2} \times s / \theta) ^ 2 + 1 = (1.96 \times 60 / 5.8) ^ 2 + 1 = 412$$

So when more than 412 process instances are simulated, we can say at a 95% confidence level that the duration falls within $\pm 0.5\%$ of the calculated mean.

14.4.2 Blocking

The evaluation based upon the $t$-distribution requires that the individual $Y_i$, $i = 1, 2, ..., n$, at least is symmetrical about the mean $\bar{Y}$ and preferably be normally distributed. This symmetry requirement can be satisfied in many simulation problems, but cannot, in general, be guaranteed. We now consider a variation of this method, known as blocking, which makes use of random variates that are always approximately normally distributed. See [Gottfried, 1984, p. 174]

The procedure is based upon the use of several consecutive short runs rather than one long run to establish a confidence interval. Specifically, consider $m$ short runs (that is, $m$ blocks), where each block contains $n$ simulated values of the performance criterion. Let

$$\bar{Y}_i = \text{the calculated mean value of the system performance criterion for each block, } i = 1, 2, ..., m.$$ (Hence, each $\bar{Y}_i$ will be averaged over $n_i$ independently generated values of the performance criterion.)

$$\bar{Y} = \text{the average of the calculated block means. Thus }$$

$$\bar{Y} = \frac{1}{m} \sum_{i=1}^{m} \bar{Y}_i$$
\( d = \) the standard deviation of the calculated block means about the average, that is,

\[
\begin{align*}
  d^2 &= \left( \frac{1}{m} \sum_{i=1}^{m} \bar{Y}_i^2 \right) - (\bar{Y})^2
\end{align*}
\]

\( \mu = \) the true mean value (unknown) of the system performance criterion.

We can again establish a confidence interval for \( \mu \) using the \( t \)-distribution. Now, however, the appropriate statistic for a given confidence lever \( (1 - \alpha) \) is

\[
- t_{m-1,1-\alpha/2} < \left( \frac{\bar{Y} - \mu}{d} \right) < t_{m-1,1-\alpha/2}
\]

or

\[
(\bar{Y} - t_{m-1,1-\alpha/2} d) < \mu < (\bar{Y} + t_{m-1,1-\alpha/2} d)
\]

This equation states that the true mean \( \mu \) falls within the interval

\[
\bar{Y} \pm t_{m-1,1-\alpha/2} d
\]

at a \( 100(1 - \alpha)\% \) confidence level.

The equations are based upon the use of block averages, which tend to be normally distributed because of the central limit theorem (see [Gottfried, 1984, p. 175]). Therefore we are more likely to satisfy the required normality condition when using one of these equations than when using

\[
\bar{Y} \pm t_{n-1,1-\alpha/2} s / \sqrt{n-1}
\]

This is the reason for favoring a blocking procedure. It should be understood, however, that the number of blocks \( m \) will usually range from ten to twenty in a typical simulation run. The normal approximation to the \( t \)-distribution cannot be used under these conditions.

**Example 12-5**

The process definition in Example 12-3 has been run two times for \( n = 100 \). Now run eight times more with the same value of \( n \), we get the following average values of 10 blocks:
Determine the limits of $\mu$ that correspond to a 95% confidence level (where $\mu$ represents the true mean value of the present worth).

The overall sample mean and the standard deviation of the block means can be obtained as

\[
\overline{Y} = \frac{1}{m} \sum_{i=1}^{m} \overline{Y}_i \\
= \frac{1}{10} (1161.59 + 1170.17 + 1141.61 + 1162.70 + 1159.21 + 1153.67 + 1163.05 + 1154.41 + 1158.35 + 1168.35) \\
= 1159.311
\]

So

\[
d^2 = \left( \frac{1}{m} \sum_{i=1}^{m} \overline{Y}_i^2 \right) - (\overline{Y})^2 \\
= \left( 1161.59^2 + 1170.17^2 + 1141.61^2 + 1162.70^2 + 1159.21^2 + 1153.67^2 + 1163.05^2 + 1154.41^2 + 1158.35^2 + 1168.35^2 \right)/10 - 1159.311^2 \\
= 60.038
\]

That is

\[
d = 7.748
\]

We have 9 degrees of freedom in this example. Therefore, the appropriated value of $t_{m-1, 1 - \alpha/2}$ can be obtained from table as $t_{9, 0.975} = 2.26$. We can now obtain the desired limits as

\[
\overline{Y} \pm t_{m-1, 1 - \alpha/2} d \\
= 1159.311 \pm (2.26)(7.748) = 1159.311 \pm 17.510
\]

Therefore

\[1141.801 < \mu < 1176.821\]
We conclude that the true mean value for the process duration falls between 1141.801 minutes and 1176.821 minutes at a 95% confidence level.

It should be noted that the equations do not explicitly involve the number of simulated values per block \( n_i \), or the standard deviation \( s_i \) of these values about each block average. The variability in the calculated block averages (and consequently \( d \)) will, however, decrease as \( n_i \) increases. This will affect the size of the confidence interval when the number of blocks \( m \) and the confidence level \((1 - \alpha)\) are fixed.

Now suppose that a simulation consisting of \( m \) blocks, with \( n \) simulated values per block, has already been carried out. We can easily calculate values for \( \bar{Y} \), \( \bar{Y} \), and \( d \), and a corresponding confidence interval, using the procedure described above. If the confidence interval is too large, then the simulation will have to be repeated (or at least restarted) using a larger number of random variates. Usually the block size \( n \) will be increased rather than the number of blocks \( m \). The new value for \( n \) can be obtained in the following manner.

Since the block averages will tend to be normally distributed, we know that their variance about the true average is given by \( \sigma^2/n \), where \( \sigma \) represents the true standard deviation of the random variates. As a rule \( \sigma \) will be unknown. We can estimate \( \sigma \), however, by utilizing the expression \( d^2 = \sigma^2/n \). To do so, we write

\[
\sigma^2 = \frac{n_1 d_1^2}{n_2 d_2^2}
\]

where \( n_1 \) represents the original (known) block size and \( d_1 \) represents the corresponding standard deviation. Thus the quantity \( n_2 d_2^2 \) can easily be obtained, where \( n_2 \) represents the new block size and \( d_2 \) represents the new standard deviation. Also \( d_2 \) can be estimated by writing

\[
T_{m-1, 1 - \alpha/2} d_2 = \theta
\]

where \( \bar{Y} \pm \theta \) is the desired confidence interval. Now the above equation can be solved directly for \( d_2 \), and \( n_2 \) can then be obtained as

\[
n_2 = \frac{n_1 d_1^2}{d_2^2}
\]

**Example 12-6**

Again consider the situation described in previous example. Determine how many **values of the present worth** will have to be determined within each block so that the true mean value falls within \( \pm 5\% \) of the calculated mean at a 95% confidence level.
If we base our calculations on the previously determined sample mean \( \bar{Y} = 1159.311 \), then

\[
\theta = 0.05 \bar{Y} = 0.05 (1159.311) = 57.96555
\]

Since \( m \) remains equal to 10, we can write

\[
t_{m-1, 1-\alpha/2} = t_{9, 0.975} = 2.26
\]

as before. Hence from \( T_{m-1, 1-\alpha/2} d_2 = \theta \), which yields

\[
d_2 = \theta / T_{m-1, 1-\alpha/2} = 57.96555 / 2.26 = 25.648
\]

We have already established that \( n_1 = 100 \), and \( d_1 = 7.748 \). Therefore

\[
n_2 = n_1 d_1^2 / d_2^2 = 100 (7.748)^2 / (25.648)^2 = 9.126
\]

We conclude that the desired confidence interval will be obtained with ten blocks provided at least 9.126 values are simulated within each block.

### 14.5 Conclusion

Collected data can be used to estimate the input variables of the simulation model and to validate a simulation model. So before using the simulation model, the system analyst should gather as much relevant data as possible. The analysis function integrated in PM can help in recording some data of an existing WfMS.

The input variables of process intercreation time, activity execution as well as delay time, work routing time, and material occupation time can be specified to be random and to be governed by a theoretical distribution function. To choose a distribution function, the special characteristics of the function, the accuracy and ease with which the function can represent a given set of empirical data, and computational efficiency to generate random variate following the function must be considered. If a set of empirical data of a random variable exists, it can be fitted via the \( \chi^2 \) test to a particular theoretical distribution function assumed to be followed by the variable.

The simulation model can be used for analyzing a WfMS and for supporting decision-making only after it is validated. To validate a simulation model, the easiest way is to compare simulation results with expected system states and performance criteria. Expected data can be obtained by theoretical prediction, hand calculation and collected historical data. If expected data cannot be obtained, common sense and knowledge from specialists will be used.
Because a WfMS is a stochastic system, the results of a simulation run are also stochastic and can not be considered as the unique system performance criteria. At a given confidence level, the mean of a system performance criterion can be evaluated via $t$-statistic to an interval around the simulated average value. To make the value interval within a certain range of percentage, the least size of the sample can be determined. If a random output variable is not symmetrically distributed, the blocking technique should be used to evaluate the simulation results of the variable.
References

[Cacutalua, 1994]

[Chorafas, 1965]

[Currid, 1994]

[Dekker, 1977]

[Eiselt, 1977]

[Fishman, 1973]

[Gerhards, 1991]

[Gordon, 1978]

[Gottfried, 1984]
REFERENCES

[Graybeal, 1980]

[Gribik, 1985]

[Harris, 1978]

[Hartley, 1975]

[Hillier, 1974]

[Hollingsworth, 1995]

[Hu, 1982]

[Kleinrock, 1996]

[Kremer, 1999]

[Koulopoulos, 1995]
[Lawrence, 1997]
Peter Lawrence: “Workflow Handbook”, 1997, John Wiley & Sons Ltd, Chichester, Great Britain

[Lehman, 1977]

[Lewis, 1979]

[Lorange, 1993]

[Maisel, 1972]

[Mitra, 1986]

[Naylor, 1966]

[Naylor, 1969]

[Riempp, 1998]

[Rosko, 1972]

[Solomon, 1983]
[Steward, 1981]

[Tijms, 1986]

[Toulemonde, 1998]

[Van der Aalst, 1998]

[Weinberg, 1980]

[WfMC, 1996]
Workflow Management Coalition: “Terminology & Glossary
http://www.wfmc.org, Brussels, Belgium

[WfMC, 1998]
Appendices

Symbols in Definitions and Algorithms

{ }: to enclose constant elements of a set or collection;
ϕ: an empty set (a set without element);
∀: for all;
∂: for any;
∃: there is, or exist;
∈: belong to;
∉: don’t belong to;
⊂: is a subset of;
⊆: is a subset or the same set of;
≡: is congruent with ;
|A|: the number of elements in set A, if A represents a set;
or absolute value of A, if A is a variable;
A ↔ B: let A have the same value as that of B;
A ∩ B: intersection of set A and set B;
A ∪ B: union of set A and set B;
A − B: remove all elements of set B from set A, if A and B represent sets;
or subtract B from A, if A and B are variables;
INT(α) or [α]: the largest integer in α or truncation of α (that is, dropping the decimals and thus retaining only the integer portion of the given quantity α);
i.e.: that is;
e.g.: for example;
call xxx: to run a procedure with underlined name (here “xxx”) and specified values or referred variables to the parameters of the procedure.
Table 1. Selected Values of the *t*-Distribution

<table>
<thead>
<tr>
<th>v</th>
<th>( t_v, 0.995 )</th>
<th>( t_v, 0.99 )</th>
<th>( t_v, 0.975 )</th>
<th>( t_v, 0.95 )</th>
<th>( t_v, 0.90 )</th>
<th>( t_v, 0.80 )</th>
<th>( t_v, 0.75 )</th>
<th>( t_v, 0.70 )</th>
<th>( t_v, 0.60 )</th>
<th>( t_v, 0.55 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>63.66</td>
<td>31.82</td>
<td>12.71</td>
<td>6.31</td>
<td>3.08</td>
<td>1.376</td>
<td>1.00</td>
<td>0.727</td>
<td>0.325</td>
<td>0.158</td>
</tr>
<tr>
<td>2</td>
<td>9.92</td>
<td>6.96</td>
<td>4.30</td>
<td>2.92</td>
<td>1.89</td>
<td>1.061</td>
<td>0.816</td>
<td>0.617</td>
<td>0.289</td>
<td>0.142</td>
</tr>
<tr>
<td>3</td>
<td>5.84</td>
<td>4.54</td>
<td>3.18</td>
<td>2.35</td>
<td>1.64</td>
<td>0.978</td>
<td>0.765</td>
<td>0.584</td>
<td>0.277</td>
<td>0.137</td>
</tr>
<tr>
<td>4</td>
<td>4.60</td>
<td>3.75</td>
<td>2.78</td>
<td>2.13</td>
<td>1.53</td>
<td>0.941</td>
<td>0.741</td>
<td>0.569</td>
<td>0.271</td>
<td>0.134</td>
</tr>
<tr>
<td>5</td>
<td>4.03</td>
<td>3.36</td>
<td>2.57</td>
<td>2.02</td>
<td>1.48</td>
<td>0.920</td>
<td>0.727</td>
<td>0.559</td>
<td>0.267</td>
<td>0.132</td>
</tr>
<tr>
<td>6</td>
<td>3.71</td>
<td>3.14</td>
<td>2.45</td>
<td>1.94</td>
<td>1.44</td>
<td>0.906</td>
<td>0.718</td>
<td>0.553</td>
<td>0.265</td>
<td>0.131</td>
</tr>
<tr>
<td>7</td>
<td>3.50</td>
<td>3.00</td>
<td>2.36</td>
<td>1.90</td>
<td>1.42</td>
<td>0.896</td>
<td>0.711</td>
<td>0.549</td>
<td>0.263</td>
<td>0.130</td>
</tr>
<tr>
<td>8</td>
<td>3.36</td>
<td>2.90</td>
<td>2.31</td>
<td>1.86</td>
<td>1.40</td>
<td>0.889</td>
<td>0.706</td>
<td>0.546</td>
<td>0.262</td>
<td>0.130</td>
</tr>
<tr>
<td>9</td>
<td>3.25</td>
<td>2.82</td>
<td>2.26</td>
<td>1.83</td>
<td>1.38</td>
<td>0.883</td>
<td>0.703</td>
<td>0.543</td>
<td>0.261</td>
<td>0.129</td>
</tr>
<tr>
<td>10</td>
<td>3.17</td>
<td>2.76</td>
<td>2.23</td>
<td>1.81</td>
<td>1.37</td>
<td>0.879</td>
<td>0.700</td>
<td>0.542</td>
<td>0.260</td>
<td>0.129</td>
</tr>
<tr>
<td>11</td>
<td>3.11</td>
<td>2.72</td>
<td>2.20</td>
<td>1.80</td>
<td>1.36</td>
<td>0.876</td>
<td>0.697</td>
<td>0.540</td>
<td>0.260</td>
<td>0.129</td>
</tr>
<tr>
<td>12</td>
<td>3.06</td>
<td>2.68</td>
<td>2.18</td>
<td>1.78</td>
<td>1.36</td>
<td>0.873</td>
<td>0.695</td>
<td>0.539</td>
<td>0.259</td>
<td>0.128</td>
</tr>
<tr>
<td>13</td>
<td>3.01</td>
<td>2.65</td>
<td>2.16</td>
<td>1.77</td>
<td>1.35</td>
<td>0.870</td>
<td>0.694</td>
<td>0.538</td>
<td>0.259</td>
<td>0.128</td>
</tr>
<tr>
<td>14</td>
<td>2.98</td>
<td>2.62</td>
<td>2.14</td>
<td>1.76</td>
<td>1.34</td>
<td>0.868</td>
<td>0.692</td>
<td>0.537</td>
<td>0.258</td>
<td>0.128</td>
</tr>
<tr>
<td>15</td>
<td>2.95</td>
<td>2.60</td>
<td>2.13</td>
<td>1.75</td>
<td>1.34</td>
<td>0.866</td>
<td>0.691</td>
<td>0.536</td>
<td>0.258</td>
<td>0.128</td>
</tr>
<tr>
<td>16</td>
<td>2.92</td>
<td>2.58</td>
<td>2.12</td>
<td>1.75</td>
<td>1.34</td>
<td>0.865</td>
<td>0.690</td>
<td>0.535</td>
<td>0.258</td>
<td>0.128</td>
</tr>
<tr>
<td>17</td>
<td>2.90</td>
<td>2.57</td>
<td>2.11</td>
<td>1.74</td>
<td>1.33</td>
<td>0.863</td>
<td>0.689</td>
<td>0.534</td>
<td>0.257</td>
<td>0.128</td>
</tr>
<tr>
<td>18</td>
<td>2.88</td>
<td>2.55</td>
<td>2.10</td>
<td>1.73</td>
<td>1.33</td>
<td>0.862</td>
<td>0.688</td>
<td>0.534</td>
<td>0.257</td>
<td>0.127</td>
</tr>
<tr>
<td>19</td>
<td>2.86</td>
<td>2.54</td>
<td>2.09</td>
<td>1.73</td>
<td>1.33</td>
<td>0.861</td>
<td>0.688</td>
<td>0.533</td>
<td>0.257</td>
<td>0.127</td>
</tr>
<tr>
<td>20</td>
<td>2.84</td>
<td>2.53</td>
<td>2.09</td>
<td>1.72</td>
<td>1.32</td>
<td>0.860</td>
<td>0.687</td>
<td>0.533</td>
<td>0.257</td>
<td>0.127</td>
</tr>
<tr>
<td>21</td>
<td>2.83</td>
<td>2.52</td>
<td>2.08</td>
<td>1.72</td>
<td>1.32</td>
<td>0.859</td>
<td>0.686</td>
<td>0.532</td>
<td>0.257</td>
<td>0.127</td>
</tr>
<tr>
<td>22</td>
<td>2.82</td>
<td>2.51</td>
<td>2.07</td>
<td>1.72</td>
<td>1.32</td>
<td>0.858</td>
<td>0.686</td>
<td>0.532</td>
<td>0.256</td>
<td>0.127</td>
</tr>
<tr>
<td>23</td>
<td>2.81</td>
<td>2.50</td>
<td>2.07</td>
<td>1.71</td>
<td>1.32</td>
<td>0.858</td>
<td>0.685</td>
<td>0.532</td>
<td>0.256</td>
<td>0.127</td>
</tr>
<tr>
<td>24</td>
<td>2.80</td>
<td>2.49</td>
<td>2.06</td>
<td>1.71</td>
<td>1.32</td>
<td>0.857</td>
<td>0.685</td>
<td>0.531</td>
<td>0.256</td>
<td>0.127</td>
</tr>
<tr>
<td>25</td>
<td>2.79</td>
<td>2.48</td>
<td>2.06</td>
<td>1.71</td>
<td>1.32</td>
<td>0.856</td>
<td>0.684</td>
<td>0.531</td>
<td>0.256</td>
<td>0.127</td>
</tr>
<tr>
<td>26</td>
<td>2.78</td>
<td>2.48</td>
<td>2.06</td>
<td>1.71</td>
<td>1.32</td>
<td>0.856</td>
<td>0.684</td>
<td>0.531</td>
<td>0.256</td>
<td>0.127</td>
</tr>
<tr>
<td>27</td>
<td>2.77</td>
<td>2.47</td>
<td>2.05</td>
<td>1.70</td>
<td>1.31</td>
<td>0.855</td>
<td>0.684</td>
<td>0.531</td>
<td>0.256</td>
<td>0.127</td>
</tr>
<tr>
<td>28</td>
<td>2.76</td>
<td>2.47</td>
<td>2.05</td>
<td>1.70</td>
<td>1.31</td>
<td>0.855</td>
<td>0.683</td>
<td>0.530</td>
<td>0.256</td>
<td>0.127</td>
</tr>
<tr>
<td>29</td>
<td>2.76</td>
<td>2.46</td>
<td>2.04</td>
<td>1.70</td>
<td>1.31</td>
<td>0.854</td>
<td>0.683</td>
<td>0.530</td>
<td>0.256</td>
<td>0.127</td>
</tr>
<tr>
<td>30</td>
<td>2.75</td>
<td>2.46</td>
<td>2.04</td>
<td>1.70</td>
<td>1.31</td>
<td>0.854</td>
<td>0.683</td>
<td>0.531</td>
<td>0.256</td>
<td>0.127</td>
</tr>
<tr>
<td>40</td>
<td>2.70</td>
<td>2.42</td>
<td>2.02</td>
<td>1.68</td>
<td>1.30</td>
<td>0.851</td>
<td>0.681</td>
<td>0.529</td>
<td>0.255</td>
<td>0.126</td>
</tr>
<tr>
<td>60</td>
<td>2.66</td>
<td>2.39</td>
<td>2.00</td>
<td>1.67</td>
<td>1.30</td>
<td>0.848</td>
<td>0.679</td>
<td>0.527</td>
<td>0.254</td>
<td>0.126</td>
</tr>
<tr>
<td>120</td>
<td>2.62</td>
<td>2.36</td>
<td>1.98</td>
<td>1.66</td>
<td>1.29</td>
<td>0.845</td>
<td>0.677</td>
<td>0.526</td>
<td>0.254</td>
<td>0.126</td>
</tr>
<tr>
<td>∞</td>
<td>2.58</td>
<td>2.33</td>
<td>1.96</td>
<td>1.645</td>
<td>1.28</td>
<td>0.842</td>
<td>0.674</td>
<td>0.524</td>
<td>0.253</td>
<td>0.126</td>
</tr>
</tbody>
</table>

\( v \): degrees of freedom

Source: [Gottfried, 1984, pp. 168-169]
260

PROCESS MODELING, VERIFICATION AND SIMULATION

Table 2. Selected Values of the N(0, 1) Distribution
Z
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
1.1
1.2
1.3
1.4
1.5
1.6
1.7
1.8
1.9
2.0
2.1
2.2
2.3
2.4
2.5
2.6
2.7
2.8
2.9
3.0
3.1
3.2
3.3
3.4
3.5
3.6
3.7
3.8
3.9
Z

0
0.0000
0.0398
0.0793
0.1179
0.1554
0.1915
0.2258
0.2580
0.2881
0.3159
0.3413
0.3643
0.3849
0.4032
0.4192
0.4332
0.4452
0.4554
0.4641
0.4713
0.4772
0.4821
0.4861
0.4893
0.4918
0.4938
0.4953
0.4965
0.4974
0.4981
0.4987
0.4990
0.4993
0.4995
0.4997
0.4998
0.4998
0.4999
0.4999
0.5000
0

1
0.0040
0.0438
0.0832
0.1217
0.1591
0.1950
0.2291
0.2612
0.2910
0.3186
0.3438
0.3665
0.3869
0.4049
0.4207
0.4345
0.4463
0.4564
0.4649
0.4719
0.4778
0.4826
0.4864
0.4896
0.4920
0.4940
0.4955
0.4966
0.4975
0.4982
0.4987
0.4991
0.4993
0.4995
0.4997
0.4998
0.4998
0.4999
0.4999
0.5000
1

2
0.0080
0.0478
0.0871
0.1255
0.1628
0.1985
0.2324
0.2642
0.2939
0.3212
0.3461
0.3686
0.3888
0.4066
0.4222
0.4357
0.4474
0.4573
0.4656
0.4726
0.4783
0.4830
0.4868
0.4898
0.4922
0.4941
0.4956
0.4967
0.4976
0.4982
0.4987
0.4991
0.4994
0.4995
0.4997
0.4998
0.4999
0.4999
0.4999
0.5000
2

3
0.0120
0.0517
0.0910
0.1293
0.1664
0.2019
0.2357
0.2673
0.2967
0.3238
0.3485
0.3708
0.3907
0.4082
0.4236
0.4370
0.4484
0.4582
0.4664
0.4732
0.4788
0.4834
0.4871
0.4901
0.4925
0.4943
0.4957
0.4968
0.4977
0.4983
0.4988
0.4991
0.4994
0.4996
0.4997
0.4998
0.4999
0.4999
0.4999
0.5000
3

4
0.0160
0.0557
0.0948
0.1331
0.1700
0.2054
0.2389
0.2704
0.2996
0.3264
0.3508
0.3729
0.3925
0.4099
0.4251
0.4382
0.4495
0.4591
0.4671
0.4738
0.4793
0.4838
0.4875
0.4904
0.4927
0.4945
0.4959
0.4969
0.4977
0.4984
0.4988
0.4992
0.4994
0.4996
0.4997
0.4998
0.4999
0.4999
0.4999
0.5000
4

Source: [Gottfried, 1984, pp. 172-173]

5
0.0199
0.0596
0.0987
0.1368
0.1736
0.2088
0.2422
0.2734
0.3023
0.3289
0.3531
0.3749
0.3944
0.4115
0.4265
0.4394
0.4505
0.4599
0.4678
0.4744
0.4798
0.4842
0.4878
0.4906
0.4929
0.4946
0.4960
0.4970
0.4978
0.4984
0.4989
0.4992
0.4994
0.4996
0.4997
0.4998
0.4999
0.4999
0.4999
0.5000
5

6
0.0239
0.0636
0.1026
0.1406
0.1772
0.2123
0.2454
0.2764
0.3051
0.3315
0.3554
0.3770
0.3962
0.4131
0.4279
0.4406
0.4515
0.4608
0.4686
0.4750
0.4803
0.4846
0.4881
0.4909
0.4931
0.4948
0.4961
0.4971
0.4979
0.4985
0.4989
0.4992
0.4994
0.4996
0.4997
0.4998
0.4999
0.4999
0.4999
0.5000
6

7
0.0279
0.0675
0.1064
0.1443
0.1808
0.2157
0.2486
0.2794
0.3078
0.3340
0.3577
0.3790
0.3980
0.4147
0.4292
0.4418
0.4525
0.4616
0.4693
0.4756
0.4808
0.4850
0.4884
0.4911
0.4932
0.4949
0.4962
0.4972
0.4979
0.4985
0.4989
0.4992
0.4995
0.4996
0.4997
0.4998
0.4999
0.4999
0.4999
0.5000
7

8
0.0319
0.0714
0.1103
0.1480
0.1844
0.2190
0.2518
0.2823
0.3106
0.3365
0.3599
0.3810
0.3997
0.4162
0.4306
0.4429
0.4535
0.4625
0.4699
0.4761
0.4812
0.4854
0.4887
0.4913
0.4934
0.4951
0.4963
0.4973
0.4980
0.4986
0.4990
0.4993
0.4995
0.4996
0.4997
0.4998
0.4999
0.4999
0.4999
0.5000
8

9
0.0359
0.0754
0.1141
0.1517
0.1879
0.2224
0.2549
0.2852
0.3133
0.3389
0.3621
0.3830
0.4015
0.4177
0.4319
0.4441
0.4545
0.4633
0.4706
0.4767
0.4817
0.4857
0.4890
0.4916
0.4936
0.4952
0.4964
0.4974
0.4981
0.4986
0.4990
0.4993
0.4995
0.4997
0.4998
0.4998
0.4999
0.4999
0.4999
0.5000
9


### Table 3. Selected Values of the $\chi^2$ Distribution

<table>
<thead>
<tr>
<th>$v$</th>
<th>$\chi^2_{v, 0.99}$</th>
<th>$\chi^2_{v, 0.95}$</th>
<th>$\chi^2_{v, 0.75}$</th>
<th>$\chi^2_{v, 0.50}$</th>
<th>$\chi^2_{v, 0.25}$</th>
<th>$\chi^2_{v, 0.05}$</th>
<th>$\chi^2_{v, 0.01}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00016</td>
<td>0.00393</td>
<td>0.1015</td>
<td>0.4549</td>
<td>1.322</td>
<td>3.841</td>
<td>6.635</td>
</tr>
<tr>
<td>2</td>
<td>0.00201</td>
<td>0.1026</td>
<td>0.5753</td>
<td>1.386</td>
<td>2.773</td>
<td>5.991</td>
<td>9.210</td>
</tr>
<tr>
<td>3</td>
<td>0.1148</td>
<td>0.3518</td>
<td>1.213</td>
<td>2.366</td>
<td>4.108</td>
<td>7.815</td>
<td>11.34</td>
</tr>
<tr>
<td>4</td>
<td>0.2971</td>
<td>0.7107</td>
<td>1.923</td>
<td>3.357</td>
<td>5.385</td>
<td>9.488</td>
<td>13.28</td>
</tr>
<tr>
<td>5</td>
<td>0.5543</td>
<td>1.1455</td>
<td>2.675</td>
<td>4.351</td>
<td>6.626</td>
<td>11.07</td>
<td>15.09</td>
</tr>
<tr>
<td>6</td>
<td>0.8720</td>
<td>1.635</td>
<td>3.455</td>
<td>5.348</td>
<td>7.841</td>
<td>12.59</td>
<td>16.81</td>
</tr>
<tr>
<td>8</td>
<td>1.646</td>
<td>2.733</td>
<td>5.071</td>
<td>7.344</td>
<td>10.22</td>
<td>15.51</td>
<td>20.09</td>
</tr>
<tr>
<td>9</td>
<td>2.088</td>
<td>3.325</td>
<td>5.899</td>
<td>8.343</td>
<td>11.39</td>
<td>16.92</td>
<td>21.67</td>
</tr>
<tr>
<td>10</td>
<td>2.558</td>
<td>3.940</td>
<td>6.737</td>
<td>9.342</td>
<td>12.55</td>
<td>18.31</td>
<td>23.21</td>
</tr>
<tr>
<td>11</td>
<td>3.053</td>
<td>4.575</td>
<td>7.584</td>
<td>10.34</td>
<td>13.70</td>
<td>19.68</td>
<td>24.73</td>
</tr>
<tr>
<td>12</td>
<td>3.571</td>
<td>5.226</td>
<td>8.438</td>
<td>11.34</td>
<td>14.84</td>
<td>21.03</td>
<td>26.22</td>
</tr>
<tr>
<td>15</td>
<td>5.229</td>
<td>7.261</td>
<td>11.04</td>
<td>14.34</td>
<td>18.25</td>
<td>25.00</td>
<td>30.58</td>
</tr>
<tr>
<td>20</td>
<td>8.260</td>
<td>10.85</td>
<td>15.45</td>
<td>19.34</td>
<td>23.83</td>
<td>31.41</td>
<td>37.57</td>
</tr>
<tr>
<td>30</td>
<td>14.95</td>
<td>18.49</td>
<td>24.48</td>
<td>29.34</td>
<td>34.80</td>
<td>43.77</td>
<td>50.89</td>
</tr>
<tr>
<td>50</td>
<td>29.71</td>
<td>34.76</td>
<td>42.94</td>
<td>49.33</td>
<td>56.33</td>
<td>67.50</td>
<td>76.15</td>
</tr>
</tbody>
</table>

$v$: degrees of freedom

Source: [Gottfried, 1984, p. 36]
### Table 4. Selected Values of the $F$-Distribution for $\alpha = 0.05$

<table>
<thead>
<tr>
<th>$F_{m,n,\alpha}$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>5</th>
<th>8</th>
<th>12</th>
<th>15</th>
<th>20</th>
<th>30</th>
<th>60</th>
<th>$\infty$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m$</td>
<td>161.4</td>
<td>199.5</td>
<td>215.7</td>
<td>230.2</td>
<td>238.9</td>
<td>243.9</td>
<td>245.9</td>
<td>248.0</td>
<td>250.1</td>
<td>252.2</td>
<td>254.3</td>
</tr>
<tr>
<td>2</td>
<td>10.13</td>
<td>9.55</td>
<td>9.28</td>
<td>9.01</td>
<td>8.85</td>
<td>8.74</td>
<td>8.70</td>
<td>8.66</td>
<td>8.62</td>
<td>8.57</td>
<td>8.53</td>
</tr>
<tr>
<td>3</td>
<td>7.71</td>
<td>6.94</td>
<td>6.59</td>
<td>6.26</td>
<td>6.04</td>
<td>5.91</td>
<td>5.86</td>
<td>5.80</td>
<td>5.75</td>
<td>5.69</td>
<td>5.63</td>
</tr>
<tr>
<td>4</td>
<td>6.61</td>
<td>5.79</td>
<td>5.41</td>
<td>5.05</td>
<td>4.82</td>
<td>4.68</td>
<td>4.62</td>
<td>4.56</td>
<td>4.50</td>
<td>4.43</td>
<td>4.36</td>
</tr>
<tr>
<td>5</td>
<td>5.99</td>
<td>5.14</td>
<td>4.76</td>
<td>4.39</td>
<td>4.15</td>
<td>4.00</td>
<td>3.94</td>
<td>3.87</td>
<td>3.81</td>
<td>3.74</td>
<td>3.67</td>
</tr>
<tr>
<td>6</td>
<td>5.59</td>
<td>4.74</td>
<td>4.35</td>
<td>3.97</td>
<td>3.73</td>
<td>3.57</td>
<td>3.51</td>
<td>3.44</td>
<td>3.38</td>
<td>3.30</td>
<td>3.23</td>
</tr>
<tr>
<td>7</td>
<td>5.32</td>
<td>4.46</td>
<td>4.07</td>
<td>3.69</td>
<td>3.44</td>
<td>3.28</td>
<td>3.22</td>
<td>3.15</td>
<td>3.08</td>
<td>3.01</td>
<td>2.93</td>
</tr>
<tr>
<td>8</td>
<td>5.12</td>
<td>4.26</td>
<td>3.86</td>
<td>3.48</td>
<td>3.23</td>
<td>3.07</td>
<td>3.01</td>
<td>2.94</td>
<td>2.86</td>
<td>2.79</td>
<td>2.71</td>
</tr>
<tr>
<td>9</td>
<td>4.96</td>
<td>4.10</td>
<td>3.71</td>
<td>3.33</td>
<td>3.07</td>
<td>2.91</td>
<td>2.85</td>
<td>2.77</td>
<td>2.70</td>
<td>2.62</td>
<td>2.54</td>
</tr>
<tr>
<td>10</td>
<td>4.84</td>
<td>3.98</td>
<td>3.59</td>
<td>3.20</td>
<td>2.95</td>
<td>2.79</td>
<td>2.72</td>
<td>2.65</td>
<td>2.57</td>
<td>2.49</td>
<td>2.40</td>
</tr>
<tr>
<td>11</td>
<td>4.75</td>
<td>3.89</td>
<td>3.49</td>
<td>3.11</td>
<td>2.85</td>
<td>2.69</td>
<td>2.62</td>
<td>2.54</td>
<td>2.47</td>
<td>2.38</td>
<td>2.30</td>
</tr>
<tr>
<td>12</td>
<td>4.67</td>
<td>3.81</td>
<td>3.41</td>
<td>3.03</td>
<td>2.77</td>
<td>2.60</td>
<td>2.55</td>
<td>2.46</td>
<td>2.38</td>
<td>2.30</td>
<td>2.21</td>
</tr>
<tr>
<td>13</td>
<td>4.60</td>
<td>3.74</td>
<td>3.34</td>
<td>2.96</td>
<td>2.70</td>
<td>2.53</td>
<td>2.46</td>
<td>2.39</td>
<td>2.31</td>
<td>2.22</td>
<td>2.13</td>
</tr>
<tr>
<td>14</td>
<td>4.54</td>
<td>3.68</td>
<td>3.29</td>
<td>2.90</td>
<td>2.64</td>
<td>2.48</td>
<td>2.40</td>
<td>2.33</td>
<td>2.25</td>
<td>2.16</td>
<td>2.07</td>
</tr>
<tr>
<td>15</td>
<td>4.49</td>
<td>3.63</td>
<td>3.24</td>
<td>2.85</td>
<td>2.59</td>
<td>2.42</td>
<td>2.35</td>
<td>2.28</td>
<td>2.20</td>
<td>2.11</td>
<td>2.01</td>
</tr>
<tr>
<td>16</td>
<td>4.45</td>
<td>3.59</td>
<td>3.20</td>
<td>2.81</td>
<td>2.55</td>
<td>2.38</td>
<td>2.31</td>
<td>2.23</td>
<td>2.15</td>
<td>2.06</td>
<td>1.96</td>
</tr>
<tr>
<td>17</td>
<td>4.41</td>
<td>3.55</td>
<td>3.16</td>
<td>2.77</td>
<td>2.51</td>
<td>2.34</td>
<td>2.27</td>
<td>2.19</td>
<td>2.11</td>
<td>2.02</td>
<td>1.92</td>
</tr>
<tr>
<td>18</td>
<td>4.38</td>
<td>3.52</td>
<td>3.13</td>
<td>2.74</td>
<td>2.48</td>
<td>2.31</td>
<td>2.23</td>
<td>2.16</td>
<td>2.07</td>
<td>1.98</td>
<td>1.88</td>
</tr>
<tr>
<td>19</td>
<td>4.35</td>
<td>3.49</td>
<td>3.10</td>
<td>2.71</td>
<td>2.45</td>
<td>2.28</td>
<td>2.20</td>
<td>2.12</td>
<td>2.04</td>
<td>1.95</td>
<td>1.85</td>
</tr>
<tr>
<td>20</td>
<td>4.32</td>
<td>3.47</td>
<td>3.07</td>
<td>2.68</td>
<td>2.42</td>
<td>2.25</td>
<td>2.18</td>
<td>2.10</td>
<td>2.01</td>
<td>1.92</td>
<td>1.82</td>
</tr>
<tr>
<td>21</td>
<td>4.30</td>
<td>3.44</td>
<td>3.05</td>
<td>2.66</td>
<td>2.40</td>
<td>2.23</td>
<td>2.15</td>
<td>2.07</td>
<td>1.98</td>
<td>1.89</td>
<td>1.79</td>
</tr>
<tr>
<td>22</td>
<td>4.28</td>
<td>3.42</td>
<td>3.03</td>
<td>2.64</td>
<td>2.37</td>
<td>2.20</td>
<td>2.13</td>
<td>2.05</td>
<td>1.96</td>
<td>1.86</td>
<td>1.76</td>
</tr>
<tr>
<td>23</td>
<td>4.26</td>
<td>3.40</td>
<td>3.01</td>
<td>2.62</td>
<td>2.36</td>
<td>2.18</td>
<td>2.11</td>
<td>2.03</td>
<td>1.94</td>
<td>1.84</td>
<td>1.73</td>
</tr>
<tr>
<td>24</td>
<td>4.24</td>
<td>3.39</td>
<td>2.99</td>
<td>2.60</td>
<td>2.34</td>
<td>2.16</td>
<td>2.09</td>
<td>2.01</td>
<td>1.92</td>
<td>1.82</td>
<td>1.71</td>
</tr>
<tr>
<td>25</td>
<td>4.23</td>
<td>3.37</td>
<td>2.98</td>
<td>2.59</td>
<td>2.32</td>
<td>2.15</td>
<td>2.07</td>
<td>1.99</td>
<td>1.90</td>
<td>1.80</td>
<td>1.69</td>
</tr>
<tr>
<td>26</td>
<td>4.21</td>
<td>3.35</td>
<td>2.96</td>
<td>2.57</td>
<td>2.31</td>
<td>2.13</td>
<td>2.06</td>
<td>1.97</td>
<td>1.88</td>
<td>1.79</td>
<td>1.67</td>
</tr>
<tr>
<td>27</td>
<td>4.20</td>
<td>3.34</td>
<td>2.95</td>
<td>2.56</td>
<td>2.29</td>
<td>2.12</td>
<td>2.04</td>
<td>1.96</td>
<td>1.87</td>
<td>1.77</td>
<td>1.65</td>
</tr>
<tr>
<td>28</td>
<td>4.18</td>
<td>3.33</td>
<td>2.93</td>
<td>2.55</td>
<td>2.28</td>
<td>2.10</td>
<td>2.03</td>
<td>1.94</td>
<td>1.85</td>
<td>1.75</td>
<td>1.64</td>
</tr>
<tr>
<td>29</td>
<td>4.17</td>
<td>3.32</td>
<td>2.92</td>
<td>2.53</td>
<td>2.27</td>
<td>2.09</td>
<td>2.01</td>
<td>1.93</td>
<td>1.84</td>
<td>1.74</td>
<td>1.62</td>
</tr>
<tr>
<td>30</td>
<td>4.08</td>
<td>3.23</td>
<td>2.84</td>
<td>2.45</td>
<td>2.18</td>
<td>2.00</td>
<td>1.92</td>
<td>1.84</td>
<td>1.74</td>
<td>1.64</td>
<td>1.51</td>
</tr>
<tr>
<td>40</td>
<td>4.00</td>
<td>3.15</td>
<td>2.76</td>
<td>2.37</td>
<td>2.10</td>
<td>1.92</td>
<td>1.84</td>
<td>1.75</td>
<td>1.65</td>
<td>1.53</td>
<td>1.39</td>
</tr>
<tr>
<td>60</td>
<td>3.92</td>
<td>3.07</td>
<td>2.68</td>
<td>2.29</td>
<td>2.02</td>
<td>1.83</td>
<td>1.75</td>
<td>1.66</td>
<td>1.55</td>
<td>1.43</td>
<td>1.25</td>
</tr>
<tr>
<td>$\infty$</td>
<td>3.84</td>
<td>3.00</td>
<td>2.60</td>
<td>2.21</td>
<td>1.94</td>
<td>1.75</td>
<td>1.67</td>
<td>1.57</td>
<td>1.46</td>
<td>1.32</td>
<td>1.00</td>
</tr>
</tbody>
</table>

$m$: degrees of freedom in the numerator

$n$: degrees of freedom in the denominator

Source: [Maisel, 1972, p. 400]
Indices

Symbolic Definitions

\[ \sigma: \text{ standard deviation, the square root of the variance} \ldots 104 \]
\[ \Gamma(n): \text{ complete gamma function} \ldots 113 \]
\[ \mu: \text{ mean, used to indicate the central tendency or location of the distribution of a random variable} \ldots 104 \]
\[ \sigma^2: \text{ Variance, used as a measure of dispersion of the value of a random variable} \ldots 104 \]
\[ (T, L, \text{Sources}(T)): \text{ a process definition} \ldots 13 \]
\[ (j, k): \text{ a link connecting activity} i \text{ to activity} k \ldots 13 \]
\[ \text{Activities}(i \rightarrow k): \text{ the set of activities on an elementary path of} i \rightarrow k \ldots 27 \]
\[ \text{Criticals}(i \rightarrow k): \text{ the set of critical activities of paths} i \rightarrow k \ldots 31 \]
\[ F(x): \text{ the cumulative distribution function for a random variable} \ldots 104 \]
\[ f(x): \text{ the probability density function of the continuous random variable} X \ldots 104 \]
\[ i \rightarrow k: \text{ activity} i \text{ has a path to activity} k \ldots 25 \]
\[ \text{Joins}(i \mid s): \text{ the set of join activities of split activity} i \text{ relevant to start activity} s \ldots 49 \]
\[ L: \text{ the set of links of a process definition} \ldots 13 \]
\[ N(\mu, \sigma): \text{ normal distribution with mean} \mu \text{ and standard deviation} \sigma \ldots 107 \]
\[ \text{Parallels}(i \rightarrow j \mid s): \text{ the set of parallel activities from split activity} i \text{ to join activity} j \text{ relevant to start activity} s \ldots 59 \]
\[ \text{Paths}(i \rightarrow k): \text{ a set of all elementary paths} i \rightarrow k \text{ in a process definition} \ldots 31 \]
\[ \text{PreJoins}(i \rightarrow q \rightarrow j \mid s): \text{ the set of ancestor join activities of a path} i \rightarrow j \text{ over activity} q \text{ relevant to start activity} s \ldots 50 \]
\[ \text{PreSplits}(i \mid s): \text{ the set of ancestors of split activity} i \text{ from start activity} s \ldots 47 \]
\[ \text{Priority}(i): \text{ join priority of activity} i \ldots 68 \]
\[ \text{Prob}(X \leq x): \text{ the probability that the value of} X \text{ for a random event does not exceed} x \ldots 107 \]
\[ ^pT(j): \text{ the set of incoming links of activity} j \ldots 14 \]
\[ \text{Reaches}(i): \text{ the set of activities which are reachable from activity} \ldots i \ldots 30 \]
\[ \text{Sinks}(T): \text{ the set of end activities of a process definition} \ldots 22 \]
\[ \text{Sources}(T): \text{ the set of start activities of a process definition} \ldots 13 \]
\[ \text{Splits}(T): \text{ the set of split activities of a process definition} \ldots 45 \]
\[ T: \text{ the set of activities of a process definition} \ldots 13 \]
ToJoins(k): the set of join activities to which activity k has a path and is reachable from the split activity of a relevant join activity ................................................................. 60
$T^S(j)$: the set of outgoing links of activity j .................................................. 14
$U(a, b)$: The uniform distribution or a rectangular distribution with values in range [a, b] ................................................................. 105
$U[a, b]$: The uniform distribution or a rectangular distribution with integer values in range [a, b] .................................................... 143
### Concept Definitions

<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1 distribution</td>
<td>112</td>
</tr>
<tr>
<td>Activity</td>
<td>2, 13</td>
</tr>
<tr>
<td>“Always” link</td>
<td>19</td>
</tr>
<tr>
<td>ancestor join activity</td>
<td>50</td>
</tr>
<tr>
<td>ancestor of split activity</td>
<td>47</td>
</tr>
<tr>
<td>Application Database</td>
<td>5</td>
</tr>
<tr>
<td>build-time</td>
<td>3</td>
</tr>
<tr>
<td>business process</td>
<td>2</td>
</tr>
<tr>
<td>cause-and-effect relationships</td>
<td>90</td>
</tr>
<tr>
<td>central limit theorem</td>
<td>108</td>
</tr>
<tr>
<td>chi-square ($\chi^2$) distribution</td>
<td>114</td>
</tr>
<tr>
<td>clock</td>
<td>96</td>
</tr>
<tr>
<td>complete gamma function</td>
<td>113</td>
</tr>
<tr>
<td>“Condition” link</td>
<td>20</td>
</tr>
<tr>
<td>confidence level</td>
<td>119</td>
</tr>
<tr>
<td>continuous random variable</td>
<td>104</td>
</tr>
<tr>
<td>critical activity</td>
<td>31</td>
</tr>
<tr>
<td>cumulative distribution function</td>
<td>104</td>
</tr>
<tr>
<td>cycle</td>
<td>35</td>
</tr>
<tr>
<td>deadlock</td>
<td>67</td>
</tr>
<tr>
<td>decision variable</td>
<td>98</td>
</tr>
<tr>
<td>delay (activity)</td>
<td>182</td>
</tr>
<tr>
<td>density function</td>
<td>104</td>
</tr>
<tr>
<td>department</td>
<td>8</td>
</tr>
<tr>
<td>destination (link)</td>
<td>13</td>
</tr>
<tr>
<td>discrete random variable</td>
<td>104</td>
</tr>
<tr>
<td>document</td>
<td>6</td>
</tr>
<tr>
<td>dynamic (system)</td>
<td>94</td>
</tr>
<tr>
<td>editor</td>
<td>176</td>
</tr>
<tr>
<td>elementary (cycle)</td>
<td>35</td>
</tr>
<tr>
<td>elementary (path)</td>
<td>27</td>
</tr>
<tr>
<td>“Else” link</td>
<td>20</td>
</tr>
<tr>
<td>empirical distribution</td>
<td>111</td>
</tr>
<tr>
<td>end activity</td>
<td>22</td>
</tr>
<tr>
<td>escalation</td>
<td>3</td>
</tr>
<tr>
<td>Espresso WFMS</td>
<td>4</td>
</tr>
<tr>
<td>event</td>
<td>95</td>
</tr>
<tr>
<td>arrival event</td>
<td>229</td>
</tr>
<tr>
<td>delay finish event</td>
<td>229</td>
</tr>
<tr>
<td>material occupation event</td>
<td>231</td>
</tr>
<tr>
<td>material release event</td>
<td>229</td>
</tr>
<tr>
<td>participant depart event</td>
<td>229</td>
</tr>
<tr>
<td>participant occupation event</td>
<td>230</td>
</tr>
<tr>
<td>process creation event</td>
<td>154</td>
</tr>
<tr>
<td>routing step event</td>
<td>229</td>
</tr>
<tr>
<td>simulation termination event</td>
<td>229</td>
</tr>
<tr>
<td>work allocation event</td>
<td>230</td>
</tr>
<tr>
<td>eventlist</td>
<td>96</td>
</tr>
<tr>
<td>“Exclusive Choice” link</td>
<td>20</td>
</tr>
<tr>
<td>exponential distribution</td>
<td>108</td>
</tr>
<tr>
<td>$F$-distribution</td>
<td>115</td>
</tr>
<tr>
<td>fixed costs</td>
<td>186</td>
</tr>
<tr>
<td>gamma distribution</td>
<td>109</td>
</tr>
<tr>
<td>heuristic model</td>
<td>94</td>
</tr>
<tr>
<td>human resource</td>
<td>175</td>
</tr>
<tr>
<td>incoming link (activity)</td>
<td>14</td>
</tr>
<tr>
<td>infinite cycle</td>
<td>36</td>
</tr>
<tr>
<td>initiator</td>
<td>178</td>
</tr>
<tr>
<td>input</td>
<td>93</td>
</tr>
<tr>
<td>instance</td>
<td>2</td>
</tr>
<tr>
<td>intercreation time (process instances)</td>
<td>99</td>
</tr>
<tr>
<td>invoked application</td>
<td>3</td>
</tr>
<tr>
<td>job</td>
<td>7</td>
</tr>
<tr>
<td>join activity</td>
<td>49</td>
</tr>
<tr>
<td>join priority</td>
<td>68</td>
</tr>
<tr>
<td>level of significance of the test</td>
<td>119</td>
</tr>
<tr>
<td>$(1 - \alpha)$</td>
<td></td>
</tr>
<tr>
<td>link</td>
<td>13</td>
</tr>
<tr>
<td>material resource</td>
<td>175</td>
</tr>
<tr>
<td>mean</td>
<td>104</td>
</tr>
<tr>
<td>model</td>
<td>94</td>
</tr>
<tr>
<td>“Multiple Choice” link</td>
<td>19</td>
</tr>
<tr>
<td>next event approach</td>
<td>95</td>
</tr>
<tr>
<td>normal approach</td>
<td>107</td>
</tr>
<tr>
<td>Notes</td>
<td>4</td>
</tr>
<tr>
<td>Notes agent</td>
<td>175</td>
</tr>
<tr>
<td>Notes group</td>
<td>8</td>
</tr>
<tr>
<td>Notes Organization Directory</td>
<td>5</td>
</tr>
<tr>
<td>operating policy</td>
<td>98</td>
</tr>
<tr>
<td>Term</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Organization Database</td>
<td>5</td>
</tr>
<tr>
<td>organizational model</td>
<td>8</td>
</tr>
<tr>
<td>organizational role</td>
<td>2</td>
</tr>
<tr>
<td>origin (link)</td>
<td>13</td>
</tr>
<tr>
<td>outgoing link (activity)</td>
<td>14</td>
</tr>
<tr>
<td>output</td>
<td>93</td>
</tr>
<tr>
<td>parallel activity</td>
<td>58</td>
</tr>
<tr>
<td>parallel routing</td>
<td>19</td>
</tr>
<tr>
<td>path</td>
<td>25</td>
</tr>
<tr>
<td>period (pseudorandom number)</td>
<td>129</td>
</tr>
<tr>
<td>person</td>
<td>8</td>
</tr>
<tr>
<td>PM</td>
<td>5</td>
</tr>
<tr>
<td>potential join activity</td>
<td>49</td>
</tr>
<tr>
<td>power of the test ((1 - \beta))</td>
<td>119</td>
</tr>
<tr>
<td>predecessor (activity)</td>
<td>13</td>
</tr>
<tr>
<td>process</td>
<td>6</td>
</tr>
<tr>
<td>Process Database</td>
<td>5</td>
</tr>
<tr>
<td>process definition</td>
<td>2, 13</td>
</tr>
<tr>
<td>process instance (process)</td>
<td>2</td>
</tr>
<tr>
<td>process protocol</td>
<td>165</td>
</tr>
<tr>
<td>ProcessModeler (PM)</td>
<td>5</td>
</tr>
<tr>
<td>pseudorandom number</td>
<td>129</td>
</tr>
<tr>
<td>random number</td>
<td>129</td>
</tr>
<tr>
<td>random variable</td>
<td>104</td>
</tr>
<tr>
<td>random variate</td>
<td>104</td>
</tr>
<tr>
<td>record</td>
<td>162</td>
</tr>
<tr>
<td>rejection probability</td>
<td>119</td>
</tr>
<tr>
<td>Replica ID</td>
<td>4</td>
</tr>
<tr>
<td>role</td>
<td>8</td>
</tr>
<tr>
<td>routing decision activity</td>
<td>20</td>
</tr>
<tr>
<td>routing option</td>
<td>19</td>
</tr>
<tr>
<td>run-time</td>
<td>3</td>
</tr>
<tr>
<td>seed (random number generator)</td>
<td>131</td>
</tr>
<tr>
<td>sequential routing</td>
<td>19</td>
</tr>
<tr>
<td>simulation</td>
<td>90</td>
</tr>
<tr>
<td>simulation model</td>
<td>95</td>
</tr>
<tr>
<td>split activity</td>
<td>45</td>
</tr>
<tr>
<td>standard deviation</td>
<td>104</td>
</tr>
<tr>
<td>standard normal distribution</td>
<td>108</td>
</tr>
<tr>
<td>standard uniform distribution</td>
<td>106</td>
</tr>
<tr>
<td>standard weekly working hours</td>
<td>96</td>
</tr>
<tr>
<td>start activity</td>
<td>14</td>
</tr>
<tr>
<td>state variable</td>
<td>99</td>
</tr>
<tr>
<td>static (system)</td>
<td>94</td>
</tr>
<tr>
<td>statistic variable</td>
<td>100</td>
</tr>
<tr>
<td>stochastic (system)</td>
<td>93</td>
</tr>
<tr>
<td>structural end activity</td>
<td>22</td>
</tr>
<tr>
<td>structural start activity</td>
<td>15</td>
</tr>
<tr>
<td>successor (activity)</td>
<td>13</td>
</tr>
<tr>
<td>system</td>
<td>93</td>
</tr>
<tr>
<td>system parameter</td>
<td>99</td>
</tr>
<tr>
<td>system performance criteria</td>
<td>102</td>
</tr>
<tr>
<td>t-distribution</td>
<td>113</td>
</tr>
<tr>
<td>task</td>
<td>6</td>
</tr>
<tr>
<td>team editor</td>
<td>179</td>
</tr>
<tr>
<td>type I error ((\alpha))</td>
<td>118</td>
</tr>
<tr>
<td>type II error ((\beta))</td>
<td>118</td>
</tr>
<tr>
<td>uniform distribution</td>
<td>105</td>
</tr>
<tr>
<td>variance</td>
<td>104</td>
</tr>
<tr>
<td>WorkForJoin</td>
<td>50</td>
</tr>
<tr>
<td>WfMS</td>
<td>2</td>
</tr>
<tr>
<td>work</td>
<td>12</td>
</tr>
<tr>
<td>work item (work)</td>
<td>3</td>
</tr>
<tr>
<td>work protocol</td>
<td>164</td>
</tr>
<tr>
<td>work record</td>
<td>162</td>
</tr>
<tr>
<td>workflow</td>
<td>2</td>
</tr>
<tr>
<td>Workflow control data</td>
<td>61</td>
</tr>
<tr>
<td>workflow management system (WfMS)</td>
<td>2</td>
</tr>
<tr>
<td>workflow participant</td>
<td>3</td>
</tr>
<tr>
<td>workgroup</td>
<td>8</td>
</tr>
<tr>
<td>worklist</td>
<td>3</td>
</tr>
</tbody>
</table>
Assumptions

Assumption 2-1. Parallel Links ................................................................. 14
Assumption 2-2. Creation of a Process Instance ....................................... 15
Assumption 2-3. Start Activities ............................................................... 15
Assumption 4-1. Activity Reachability ...................................................... 31
Assumption 4-2. Infinite Cycle .................................................................. 36
Assumption 5-1. Synchronization at a Join Activity .................................... 50
Assumption 5-2. Join List (Run-time) ....................................................... 60
Assumption 6-1. Deadlock Release ........................................................... 68
Assumption 11-1. Simulating Process Creations ........................................ 155
Assumption 11-2. Simulating a “Multiple Choice” Link ............................. 157
Assumption 11-3. Simulating an Activity-dependent Variable .................... 159
Assumption 11-4. Simulating a Variable with Multiple Values ................... 159
Assumption 11-5. Simulating Stop of a Process Work ............................... 160
Assumption 12-1. Simulating a Notes Agent ............................................. 177
Assumption 12-2. No Simulation of a Stand-in .......................................... 178
Assumption 12-3. Simulating Editor “Computed” ..................................... 178
Assumption 12-4. Simulating Workflow Participants of the Predecessors .... 179
Assumption 12-5. Simulating Completion of an Activity ............................ 180
Assumption 12-6. Simulating Workflow Participants among a Team .......... 180
Assumption 12-7. Simulating Behavior of a Waiting Queue ....................... 181
Assumption 12-8. Simulating the Waiting Queue of a Team Member .......... 182
Assumption 12-9. Distribution of Execution/Delay Time of an Activity ....... 183
Assumption 12-10. Continuous Activity Execution ...................................... 184
Assumption 12-11. Simulating Delay of an Activity ................................... 184
Assumption 12-12. Standard Deviation of Material Occupation Time .......... 185
Assumption 12-13. Continuous Occupation of a Material ............................ 185
Assumption 12-14. Simulating Waiting Queue for Material Occupation ....... 186
Assumption 12-15. Simulating Multiple Workflow Participants of an Activity ................................................................. 187
Assumption 12-16. Simulating Material Occupation by Multiple Participants ................................................................. 187
Assumption 12-17. Dividing Work among Multiple Workflow Participants 188
Assumption 12-18. Dummy Participant of an Editor .................................... 189
Assumption 12-21. Work Calendar ............................................................ 193
Assumption 12-22. Effect of Participant Settings ........................................ 193
Assumption 12-23. Common Public Holidays .......................................... 198
Assumption 13-1. Queuing Rule .............................................................. 215
## Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adding Work to Queue</td>
<td>203</td>
</tr>
<tr>
<td>Checking an Activity Split in a Protocol</td>
<td>168</td>
</tr>
<tr>
<td>Connecting a Parallel Work</td>
<td>165</td>
</tr>
<tr>
<td>Copying a Work Protocol</td>
<td>167</td>
</tr>
<tr>
<td>Detecting Deadlocks</td>
<td>82</td>
</tr>
<tr>
<td>Determining Activity Reachable to End</td>
<td>36</td>
</tr>
<tr>
<td>Determining Joins</td>
<td>57</td>
</tr>
<tr>
<td>Determining Potential Start Activities</td>
<td>42</td>
</tr>
<tr>
<td>Determining Reachability and Criticality</td>
<td>32</td>
</tr>
<tr>
<td>Determining Workflow Control Data</td>
<td>62</td>
</tr>
<tr>
<td>Disconnecting a Parallel Work</td>
<td>166</td>
</tr>
<tr>
<td>Drawing Gamma Distribution</td>
<td>111</td>
</tr>
<tr>
<td>Eliminating a Work Record</td>
<td>170</td>
</tr>
<tr>
<td>Eliminating Protocol Records</td>
<td>167</td>
</tr>
<tr>
<td>Finding a Path</td>
<td>28</td>
</tr>
<tr>
<td>Generating an Empirical Variate</td>
<td>149</td>
</tr>
<tr>
<td>Generating an Exponential Variate</td>
<td>145</td>
</tr>
<tr>
<td>Generating a Gamma Variate</td>
<td>146</td>
</tr>
<tr>
<td>Generating a Gamma Variate with Non-Integer Valued Shape Parameter</td>
<td>147</td>
</tr>
<tr>
<td>Generating a Normal Variate</td>
<td>144</td>
</tr>
<tr>
<td>Generating Random Numbers</td>
<td>132</td>
</tr>
<tr>
<td>Getting All Split Activities in Protocols</td>
<td>169</td>
</tr>
<tr>
<td>Getting Ancestor Joins</td>
<td>57</td>
</tr>
<tr>
<td>Getting Ancestors of a Split Activity</td>
<td>48</td>
</tr>
<tr>
<td>Getting Process Critical Paths</td>
<td>172</td>
</tr>
<tr>
<td>Getting Elementary Cycles</td>
<td>39</td>
</tr>
<tr>
<td>Getting Elementary Deadlocks</td>
<td>69</td>
</tr>
<tr>
<td>Getting Infinite Elementary Cycles</td>
<td>40</td>
</tr>
<tr>
<td>Getting Invalid IT Resource Activities</td>
<td>22</td>
</tr>
<tr>
<td>Getting One Deadlock</td>
<td>78</td>
</tr>
<tr>
<td>Getting Parallel Waiting Work Items</td>
<td>81</td>
</tr>
<tr>
<td>Getting the First Split Activity</td>
<td>61</td>
</tr>
<tr>
<td>Grouping Deadlocks</td>
<td>71</td>
</tr>
<tr>
<td>Keeping a Process Definition</td>
<td>16</td>
</tr>
<tr>
<td>Next-Event Approach</td>
<td>96</td>
</tr>
<tr>
<td>Releasing a Deadlock</td>
<td>82</td>
</tr>
<tr>
<td>Routing Work to Activity</td>
<td>80</td>
</tr>
<tr>
<td>Updating Split Activities</td>
<td>46</td>
</tr>
</tbody>
</table>
Figures

Figure 1-1. Relationships between Basic Workflow Concepts .................. 1
Figure 1-2. Espresso Databases and PM Functions ................................... 6
Figure 1-3. Process Definition “Order” .................................................. 9
Figure 1-4. Process Definition “Report” ................................................... 10
Figure 1-5. Process Definition “Loan” ...................................................... 11
Figure 2-1. Link .................................................................................. 13
Figure 2-2. Process definition ................................................................. 13
Figure 2-3. Start Activity ..................................................................... 15
Figure 2-4. Keeping a Process Definition ............................................... 18
Figure 3-1. Routing Options ................................................................. 21
Figure 3-2. Display of Routing Options .................................................. 21
Figure 3-3. Structural End Activities 2, 4, 5, 6 and 7 ............................. 23
Figure 3-4. Getting End and Invalid IT Resource Activities ..................... 24
Figure 3-5. Start Activity ................................................................. 26
Figure 3-6. Cycle ................................................................................. 41
Figure 3-7. Verification Results—Infinite Cycle ..................................... 41
Figure 3-8. Infinite Cycles ................................................................... 41
Figure 4-1. Potential Start Activities ...................................................... 43
Figure 4-2. Verification Result—Start Activity ........................................ 43
Figure 4-3. Getting Potential Start Activities, Paths, and Infinite Cycles .. 44
Figure 4-4. Split Activity ..................................................................... 45
Figure 4-5. Not a Split Activity—Exclusive Links .................................... 46
Figure 4-6. Not a Split Activity—Else Link ............................................. 46
Figure 4-7. Ancestor of Split Activity ...................................................... 48
Figure 4-8. Split and Join ..................................................................... 51
Figure 4-9. Multi-Split and Join .............................................................. 52
Figure 4-10. Not a Join—without Split .................................................... 52
Figure 4-11. Not a Join—Critical on Path ................................................ 53
Figure 4-12. Not a Join—Itself on Path .................................................... 54
Figure 4-13. Join at Split ..................................................................... 54
Figure 4-14. Join—Start on Path ............................................................. 55
Figure 4-15. Join—Ancestor on Path ....................................................... 55
Figure 4-16. Not a Join—Ancestor Join ................................................... 56
Figure 4-17. Parallel Activity ................................................................. 59
Figure 4-18. To-Join List ..................................................................... 60
Figure 4-19. Workflow Control Data ...................................................... 64
Figure 4-20. Workflow control Data—Change Start ............................... 65
Figure 4-21. Determining Workflow Control Data ................................. 66
Figure 4-22. Deadlock ....................................................................... 67
Figure 4-23. Join Priority Specification .................................................. 69
Figure 4-24. Group Deadlocks ............................................................... 72
Figure 4-25. Deadlock Release ............................................................... 73
Figure 4-26. Deadlock Release: 3<4<6 .................................................... 74
Figure 6-6. Deadlock Release: 3<6<4 ................................................. 75
Figure 6-7. Deadlock Release: 4<3<6 ............................................... 76
Figure 6-8. Deadlock Release: 4<6<3 .............................................. 77
Figure 6-9. Deadlock Release: 6<3<4 ............................................... 77
Figure 6-10. Deadlock Release: 6<4<3 ............................................. 78
Figure 6-11. Detect and Release Deadlocks ........................................ 84
Figure 7-1. Summary of Process Definitions and Algorithms ............... 85
Figure 7-2. Deadlock for Synchronization ......................................... 87
Figure 8-1. System and Categorization ............................................ 93
Figure 8-2. Simulation Model and Variables ...................................... 97
Figure 8-3. Calculation of Time Integrated Value ................................ 101
Figure 9-1. Uniform Distribution with that $\mu = 5$ and $\sigma = 2$ .......... 106
Figure 9-2. Normal Distribution with that $\mu = 5$ and $\sigma = 2$ .......... 107
Figure 9-3. Exponential Distribution with that $\mu = 5$ ....................... 109
Figure 9-4. Gamma Distribution with that $\mu = 5$ and $\sigma = 2$ ............ 110
Figure 9-5. Empirical Density Function ............................................. 111
Figure 9-6. Empirical Cumulative Distribution Function ....................... 112
Figure 9-7. $t$-Distribution and $N(0, 1)$ Distribution .......................... 113
Figure 9-8. $\chi^2$ Distribution with Different Freedom $n$ ...................... 114
Figure 9-9. A Power Curve ............................................................... 119
Figure 9-10. The Critical Region of $t$-Distribution ............................. 120
Figure 9-11. The Critical Region of $\chi^2$ Distribution ......................... 122
Figure 9-12. Relative Frequency of Observed Intercreation Time .......... 124
Figure 9-13. Estimated Distribution of Process Intercreation Time .......... 125
Figure 10-1. The Inverse Transformation Method .................................. 141
Figure 10-2. The Rejection Method .................................................... 142
Figure 10-3. $U(0, 1)$ Cumulative Distribution ................................... 143
Figure 11-1. Process Life Period Settings ......................................... 153
Figure 11-2. Process Creation Settings ............................................. 154
Figure 11-3. Process Routing Choice Settings .................................... 157
Figure 11-4. Process Variable Settings ............................................. 158
Figure 11-5. Graphical Process States .............................................. 161
Figure 11-6. Parallel Work Record Loop .......................................... 162
Figure 11-7. Parallel Work Record Loop—Add .................................... 163
Figure 11-8. Parallel Work Record Loop—Remove ................................ 163
Figure 11-9. Work Protocol Records ................................................ 164
Figure 11-10. Relationships between Algorithms of Simulated Work Items 165
Figure 11-11. Process Definition—Eliminating a Work Record ............... 170
Figure 11-12. Eliminate a Parallel Work Record—before ...................... 171
Figure 11-13. Eliminate a Parallel Work Record—aft er .......................... 171
Figure 12-1. Activity Participant Definition ........................................ 176
Figure 12-2. Editor, Organizational Roles and Workflow Participants ....... 177
Figure 12-3. Activity Resource Definition ......................................... 183
Figure 12-4. Distribution Function Specification ................................... 191
Figure 12-5. Participant Specification ............................................... 192
Figure 12-6. Participating Settings of Team Parameters ............................. 194
Figure 12-7. Assignment Specification .................................................. 195
Figure 12-8. Assignment Specification—Team Parameters ......................... 196
Figure 12-9. Public Holiday Specification .............................................. 197
Figure 12-10. Default Public Holiday Specification .................................. 197
Figure 12-11. Simulated Resources ...................................................... 198
Figure 12-12. Graphic Resource States .................................................. 199
Figure 12-13. Dynamic Resource Window ............................................. 200
Figure 12-14. Queue Records ............................................................... 201
Figure 12-15. Queue Records—First-in-First-out ..................................... 202
Figure 12-16. Queue Records—Last-in-First-out ..................................... 202
Figure 12-17. Queue Records—in Shortest Execution Time ......................... 202
Figure 13-1. Choose Creator of a Process Instance ..................................... 207
Figure 13-2. Choose Workflow Participants of an Activity ......................... 208
Figure 13-3. Choose Variate of a Variable ............................................. 208
Figure 13-4. Choose Routing Links ....................................................... 209
Figure 13-5. Simulation Beginning/Ending Settings .................................... 212
Figure 13-6. Process Protocol Settings .................................................. 213
Figure 13-7. Database Configurations for Simulation .................................. 214
Figure 13-8. Worklist Sorting Settings ................................................... 215
Figure 13-9. Simulation View Overall Settings ......................................... 216
Figure 13-10. Simulation Sound Settings ............................................... 217
Figure 13-11. Simulation View Process Map Settings ................................ 217
Figure 13-12. Process Protocol .............................................................. 219
Figure 13-13. Process Definition and Process Protocol ................................ 220
Figure 13-14. Simulation Summary Report .............................................. 221
Figure 13-15. Activity Report ............................................................... 222
Figure 13-16. Resource Report ............................................................... 225
Figure 13-17. Resource Report—Chart .................................................. 225
Figure 13-18. Work Allocation Report ..................................................... 226
Figure 13-19. Work Allocation Report—Resource-Oriented ......................... 227
Figure 13-20. Work Allocation Report—Activity-Oriented ......................... 227
Figure 13-21. Events and Relevant Algorithms and States .......................... 231
Figure 14-1. Process Instances Associated with a Process Definition .............. 237
Figure 14-2. Collect Duration of Completed Process Instances ..................... 238
Figure 14-3. Single-channel Single-station Queue System .......................... 242
Figure 14-4. Simulation Results of a M/G/1 Queue .................................... 244
## Examples

| Example 1-1 | 9 | Example 5-11 | 54 |
| Example 1-2 | 10 | Example 5-12 | 55 |
| Example 1-3 | 10 | Example 5-13 | 55 |
| Example 2-1 | 13 | Example 5-14 | 59 |
| Example 2-2 | 14 | Example 5-15 | 60 |
| Example 3-1 | 23 | Example 5-16 | 64 |
| Example 4-1 | 26 | Example 6-1 | 67 |
| Example 4-2 | 30 | Example 6-2 | 68 |
| Example 4-3 | 31 | Example 6-3 | 72 |
| Example 4-4 | 33 | Example 10-1 | 132 |
| Example 4-5 | 35 | Example 10-2 | 133 |
| Example 4-6 | 41 | Example 10-3* | 134 |
| Example 4-7 | 43 | Example 10-4 | 135 |
| Example 5-1 | 45 | Example 10-5 | 137 |
| Example 5-2 | 46 | Example 11-1 | 155 |
| Example 5-3 | 46 | Example 11-2 | 163 |
| Example 5-4 | 48 | Example 11-3 | 164 |
| Example 5-5 | 51 | Example 12-1 | 243 |
| Example 5-6 | 52 | Example 12-2 | 246 |
| Example 5-7 | 52 | Example 12-3 | 247 |
| Example 5-8 | 53 | Example 12-4 | 248 |
| Example 5-9 | 53 | Example 12-5 | 250 |
| Example 5-10 | 54 | Example 12-6 | 252 |