Process Design, 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 (Dr. rer. pol.)

Under the supervision of professor Ludwig Nastansky

at the

University of Paderborn
Germany

2001

ACKNOWLEDGEMENTS

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

I am very grateful to Professor Dr. Ludwig Nastansky, my mentor and principal supervisor, who helped me with the dissertation in many different ways, particularly who offered me a job at the university during this time so that I could work for PAVONE to gather necessary experience and to implement the theory and methods discussed in the dissertation in the ProcessModeler product. I also wish to express my gratitude to Professor Dr. Leena Suhl, the second supervisor of the dissertation, and other members of the dissertation committee, Professor Dr. Wilhelm Dangelmaier and Professor Dr. Manfred Kraft.

I am indebted to Mr. Howard Almond, who revised the whole work and provided me with many suggestions and diverse help during my doctoral work, and Dr. Rolf Kremer, who kindly read a preliminary draft and provided me with his critical comments and various advices. 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 for his revision of this work, and Mr. Jens Winkelmann for his ideas and for 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 at PAVONE and its partner firms for offering me a lot of practical suggestions for improving the PM product 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 with the product devised by this team that I began my doctoral work.

Finally, I wish to thank my mother, who never forgot to bless me for a happy, successful life.

Hong Zhang
Paderborn, August 2001

CONTENTS

*: not implemented in the Espresso ProcessModeler

PREFACE

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]

Compared to the high efficiencies that have been achieved in automated manufacturing processes, the efficiencies associated with management and business processes involving people remain 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 (IT), especially the appearance and enhancement of groupware techniques, makes 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/Gabathuler/Jansen/Rossini /Wylie/Schaper, 1998, p. 22]

Most organizations using WfMS are motivated by the 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 provided 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 applied in a practical visual modeling tool—ProcessModeler (PM), for design, verification and simulation of process definitions implemented in a WfMS.

"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 easily modeled as a graph. In accordance with the process definition, the business processes can then be automatically created at one of the start activities and be routed along the links from activity to activity. The diverse routing options of links allow the activity execution thread of a business process to be dynamically determined according to the real world circumstances.

The activity network of a process definition can be simply and flexibly designed with PM and can, thus, be very complex. However all the activities within a process definition must be reachable from a start activity. No infinite cycle in a process definition is allowed. According to the network structure, PM determines for the workflow engine join activities and other workflow control data of a process definition. At a join activity, parallel activity execution threads will be merged into one thread. Work items waiting for joining at the join activities may at run-time yield a deadlock situation of a business process. Therefore, PM detects the deadlock cycles of a process definition and assigns join priorities to the join activities so that the workflow engine can release deadlock situations.

Uneven parallel activity execution threads in a process definition can prolong the duration of a business process. Shortage or unbalanced allocation of human and material resources demanded for the execution of the activities can also lengthen the duration. "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/ter Hofstede, 1998, p. 17]

Simulation is to make experiments on a simulation model (simulator) that 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].

To simulate a WfMS, a stochastic and dynamic system, a broad body of input data should be estimated upon the collected data or guessed by specialists so that they represent the features of the real world system as well as possible. The validity of the results obtained 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 should be validated, or shown to actually represent the system being studied. Therefore, the theory and methods requisite for the proper development and operation of the simulator are presented in this work.

Some algorithms for process definition and especially simulation are built upon assumptions. The assumptions specify prerequisites, constraints, or principles for process design and simulation. They should not diverge from the behavior of the real world business processes.

The algorithms 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

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

Part Two covers the basic simulation knowledge and methods that are needed in the simulation study. The goals of the part are to outline the basic concepts of the simulation and the simulation model (Chapter 8), 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 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 and the simulation of processes. Chapter 12 is devoted to the resource simulation 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 carried out 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].

The basic concepts with the relationships illustrated in Figure 1-1 are given by the Workflow Management Coalition.

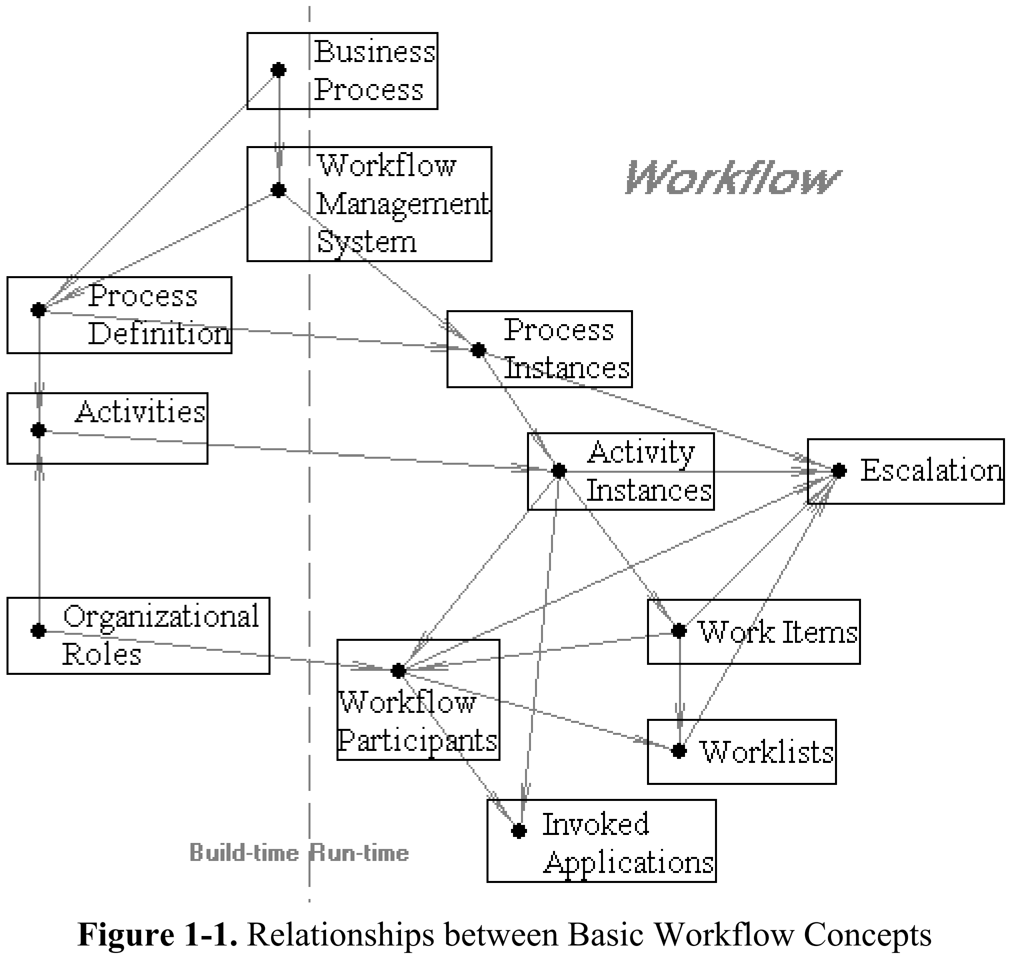


Figure 1-1. Relationships between Basic Workflow Concepts

- *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 an 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 (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 affect those belonging to run-time.

1.2 The Espresso Workflow Management System

“Workflow management targets a severe reduction of paper worked on inside offices, mailed between organizations, copied and carried around both by internal personnel as well as by external company representatives. All the required logistics to select necessary material and the costs involved are to be reduced.” [Hilpert/Riempp/Nastansky, 1994, p.1]

In 1994, based upon two years of research by the GCC (originally, the CSDS project group) of the University of Paderborn, PAVONE began developing the Espresso WfMS, originally called the GroupFlow System. “GroupFlow has been implemented using Lotus Notes as the basic development platform and underlying distributed architecture. The user interfaces on the client sides are either based on Notes-native FORM and VIEW concepts, or developed using several other graphical fronted tools when appropriate for the respective user tasks to be performed. On the backend server side of GroupFlow, solely Notes technology has been used for data repositories of the actual business information content, for the workflow structure parts, and the various workflow runtime engines supporting processes like messaging, replication, event management or gateway connections.” [Nastansky/Hilpert, 1994, p.1]

“A sustainable part of work in an office environment involves a combination of highly structured processes and tasks where the process is fuzzy and the rules, routes and roles are dynamically defined as the work is being done, This is why workflow systems alone are not as successful as expected and deemed to be ‘too rigid’.” [Huth/Erdmann/Nastansky, 2001, p. 2] “Thus, an essential challenge for workflow systems is to support structure and flexibility equally. On the one hand, repeatedly recurring standard workflows and on the other hand ad-hoc workflows or exception handling must be modeled and supported.” [Ott/Nastansky/Brockmeyer, 1996, p.3]

The *Espresso WfMS* is a document-oriented workflow application system that supports well-structured workflow processes as well as flexible and loosely structured processes—routing option specification, exception handling of routing and ad-hoc workflow are allowed in the system (see [Nastansky /Hilpert, 1994]). 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/Gabathuler/Jansen/Rossini/Wylie/Schaper, 1998, p.19]

In the application database of the Espresso WfMS, Notes build-in workflow mechanisms are utilized by the Espresso workflow engine 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 data 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 activity of the process instance. A Notes view can be designed for presenting the worklist of a workflow participant. Scheduled or mail-triggered Notes agents can be assigned to 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 thus 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 guarantee 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 kinds of Lotus Notes databases:

- an application database integrated with the Espresso workflow engine, called *Espresso application database*, for automation of the business processes within the database;
- a *PAVONE 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
- an *Espresso Process Database* holding the process definitions.

For a simulation study, the Espresso Simulation Database is required for saving and retrieving simulation settings and simulation reports.

1.2.1 The ProcessModeler

The *ProcessModeler* (PM) is a visual modeling tool for process designing, simulating and analyzing in the Espresso WfMS.

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 dialogue 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 design errors, such as non-reachable activities, infinite loops, deadlock cycles, not available resources, etc., and generate workflow control data for the workflow engine.

A process definition is stored in an Espresso Process Database. In the database, every process definition consists of a header document, a layout 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 the incoming activities. Once completed in PM, a process definition can be immediately implemented in an Espresso application database.

PM allows the process instances of process definitions to be simulated or animated. Thus, behaviors of the process instances in accordance with the process definitions can be forecasted and potential bottlenecks can be detected, before the process definitions are implemented in the Espresso WfMS. For a process definition that has been implemented in the Espresso application database, the analyzer, integrated in PM, can read running process instances of a process definition for supervising and controlling flows of the work items in the WfMS.

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 clusters (task 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] However, “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]

A process definition can also be saved as an *XML* (Extensible Markup Language) file, so that PM can, for example, simulate a process definition modeled by another tool without simulation model, or be utilized to design

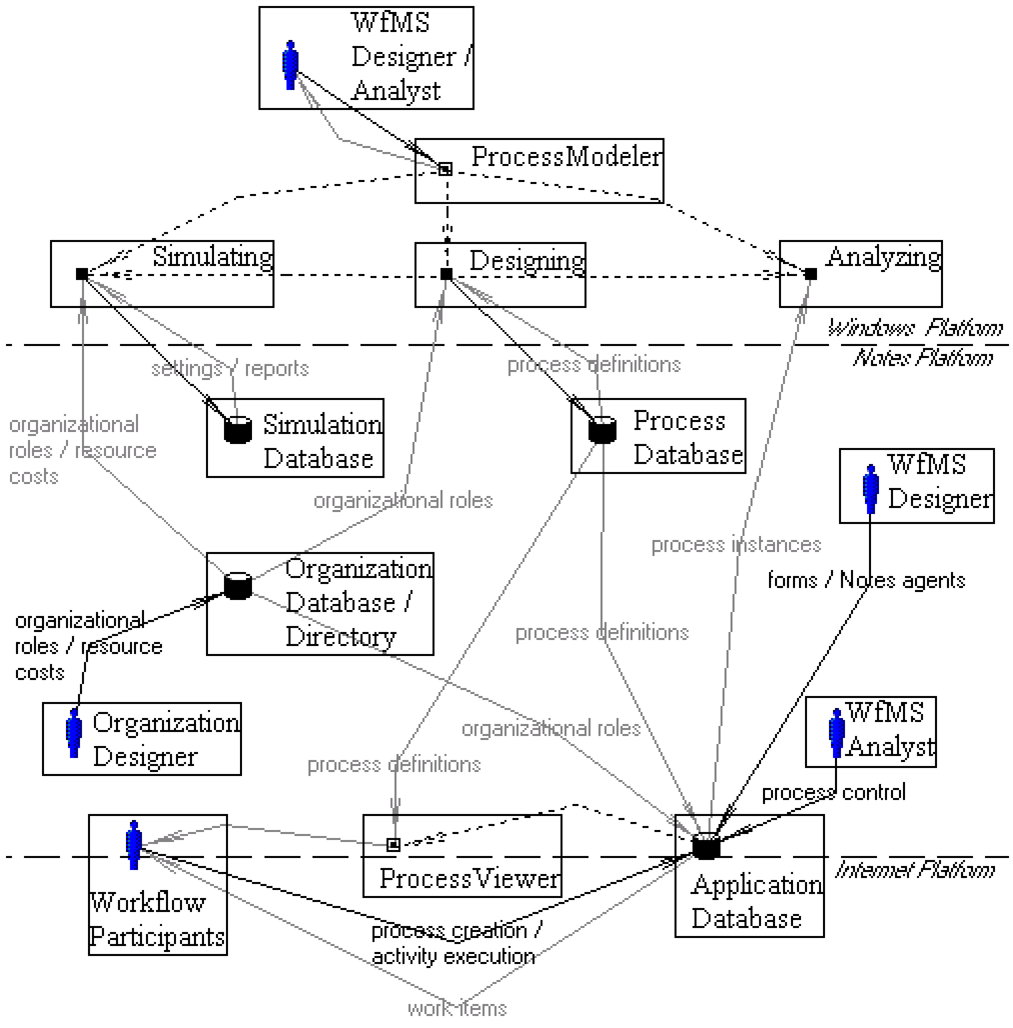


Figure 1-2. Espresso Databases and PM Functions

process definitions for another WfMS. See Appendix of DTD of XML for PAVONE Process Definition.

1.2.2 The Application Database

One or multiple application databases integrated with the Espresso workflow engine (called PAVONE Process Engine) act as the run-time environment of the Espresso WfMS. The Espresso 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 an Espresso Process Database or the ad-hoc workflows defined/saved at run-time in the Espresso application databases. 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 Espresso 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 by a workflow participant belonging to the organizational role assigned to a start activity of a process definition implemented in an Espresso 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.

A Notes document representing the instance of an activity is allocated to the worklist of every workflow participant belonging to the organizational role assigned to the activity. A human workflow participant can open a document in his worklist, in a Notes client or via an Internet web browser, and execute the activity. The description of the activity is displayed in a specified Notes form that filters certain data of the document, and the workflow participant may fill out the form as required. According to specific circumstances of a business process, the workflow participants can add additional activities via defining or loading an ad-hoc workflow, or alter the routing of the document (inside or outside the associated process definition) via exception handling.

Sometimes it is necessary to automatically invoke other applications in order to give the current workflow participant the information and tools required for the execution of an activity. The ProcessViewer integrated in the Espresso application database can be invoked by the workflow participants. It graphically depicts how the document has been routed from activity to activity and from person to person in accordance with one or more process definitions and ad-hoc workflows, and it shows the activities yet to come. The user can also choose to view an animation of the routing of the document.

If a Notes agent is specified as a workflow participant, it will execute the activity at a scheduled time or when a mail comes in.

When an activity within a process instance is completed and then the work associated with the process instance flows from this activity to the next according to the connections (links) 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 via e-mail to the workflow participants of the next activity, but is allocated to their worklists. However, 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 an activity is completed and the work will flow to multiple next activities, the document representing the instance of the activity as well as the process instance will be copied in order to represent respectively each of the next activities. In this case, there are parallel work items associated with the same process instance in the Espresso application database. Before the work at a join activity can be executed or completed, the documents that are associated with the same process instance and represent different instances of the previous activities will be joined into one document for representing the instance of the join activity.

1.2.3 The Organizational Model

A PAVONE Organization Database and/or a Notes Organization Directory comprise an organizational model for the Espresso WfMS (see [Ott, 1999] for the design of the organization database). 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.” [WfMC, 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 a PAVONE 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. A person can belong to different Notes group.
- *Workgroup*: a team of people defined in a PAVONE Organization Database for a certain organizational objective or project. A person can belong to different workgroups. A member can be specified as the manager of a workgroup.
- *Department*: a team of people defined in a PAVONE 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 one and only one department in the organization database. That is, departments in the 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 for doing a kind of work. Roles are defined in a PAVONE Organization Database. A person can belong to different roles. Role parameters can be determined at run-time for allocating a work associated with the same activity to different workflow participants under different circumstances.

A PAVONE Organization Database and/or a Notes Organization Directory must be configured to the Espresso application database so that workflow participants for executing an activity within a process instance can be determined at run-time by the Espresso workflow engine. At process build-time, the databases can be used by PM to assign defined organizational roles to the activities of a process definition.

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/Kortanek, 1985] and [Steward, 1981]). Activities are represented by icons and connected by directed arcs, called links in PM. Links indicate the flows of work items between activities and affect the execution thread of a process instance—parallel or sequential.

Example 1-1. Process Definition and Instances

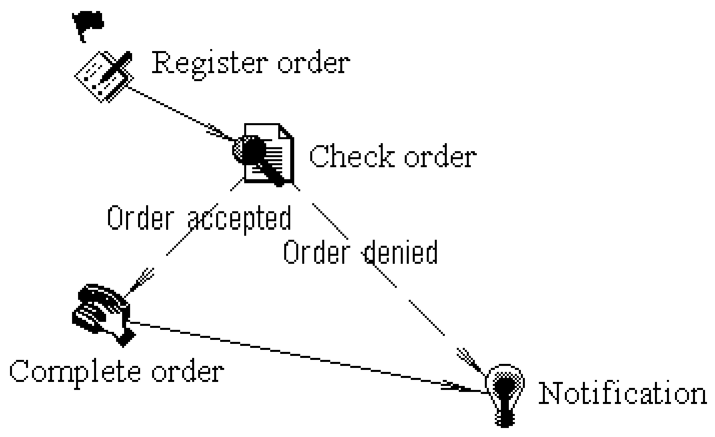


Figure 1-3. Process Definition "Order"

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).

In a WfMS, a process instance will be 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.

An activity execution thread of a process instance generated at run-time in accordance with the process definition can be either

- Register order → Check order → Notification; or
- Register order → Check order → Complete order → Notification.

Example 1-2. Cycle in Process Definition

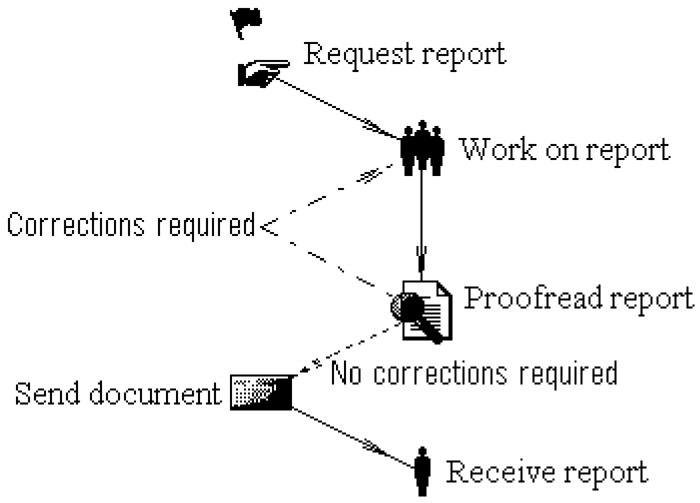


Figure 1-4. Process Definition “Report”

Process definition “Report” presented in Figure 1-4 contains five activities. The 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 activity execution threads of

- Request report → Work on report → Proofread report → Send document → Receive report,
- Request report → Work on report → Proofread report → Work on report → Proofread report → Send document → Receive report, or
- Request report → Work on report → Proofread report → Work on report → Proofread report → Work on report → Proofread report → Send document → Receive report, etc.

Example 1-3. Split and Join

Process definition “Loan” shown in Figure 1-5 consists of seven activities. In this process definition, if the 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 personal creditworthiness” and “Check Asset valuations” are hence called parallel activities.

Because after execution of activity “Evaluate”, a single execution thread may split into two parallel execution threads, activity “Evaluate” is thus a split activity. Before execution of activity “Approve credit”,

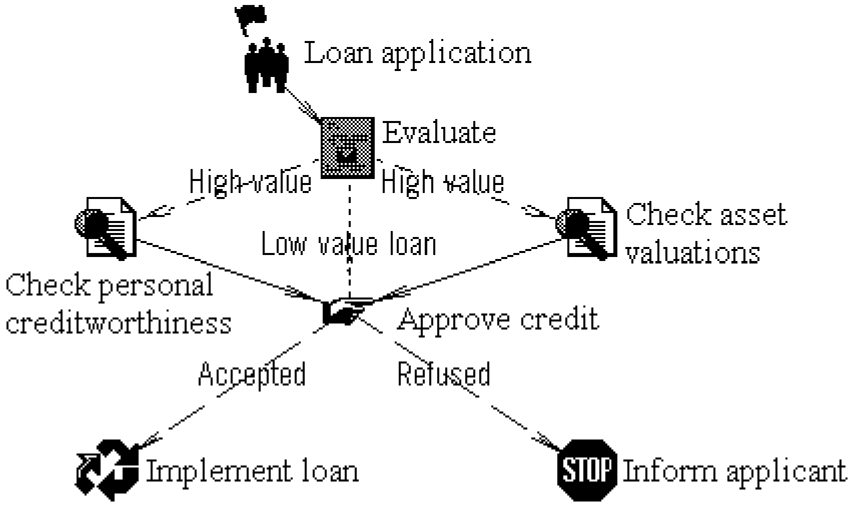


Figure 1-5. Process Definition “Loan”

multiple concurrent execution threads of a process instance converge into a single execution thread. Therefore, activity “Approve credit” is a join activity. Before execution of a join activity, synchronization may be required for joining parallel work items coming 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.

A WfMS enables the automation of business processes. A process instance is an automated business process that is created and routed mainly in accordance with a process definition, a network of activities. An activity within a process definition is assigned to an organization role, and an instance of the activity within a process instance is executed by workflow participants that are resolved at run-time from the organization role. For each workflow participant, there is a worklist for allocating the work items associated with activity instances. Thus, in a WfMS, a work item associated with a process instance flows automatically in accordance with a process definition, from activity to activity, from workflow participant to workflow participant.

The ProcessModeler (PM), part of the Espresso WfMS, is a product where the methodology discussed in this work are implemented, except those sections explicitly marked with asterisk (“*”).

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*.

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

2 PROCESS DEFINITION

A *process definition*, denoted by (T, L, Sources(T)), consists of a finite and non-empty set of activities, denoted by T, $T \neq \phi$, a collection of links connecting certain pairs of activities, denoted by L, and a non-empty set of start activities, denoted by *Sources*(T). 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 executed by human and/or IT resources. 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

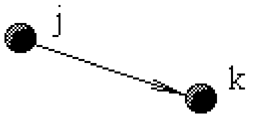


Figure 2-1. Link

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

Example 2-1. Activity/Link Set

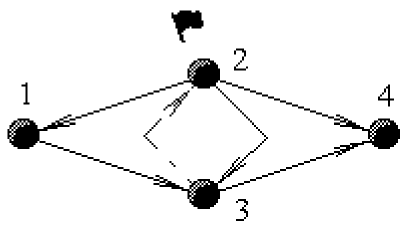


Figure 2-2. Process Definition

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

$T = \{1, 2, 3, 4\}$

and six links, i.e.

$L = \{(1, 3), (2, 1), (2, 3), (2, 4), (3, 2), (3, 4)\}$

Parallel links are not allowed 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.

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 $i \neq k$.

The set of *outgoing link*s of activity j, denoted by $T^S(j)$, is defined as

$\{(j, k) \mid \forall k\in T \text{ with } (j, k)\in L\}$

The set of *incoming link*s of activity j, denoted by ${}^PT(j)$, is defined as

$\{(k, j) \mid \forall k\in T \text{ with } (k, j)\in L\}$

Example 2-2. Set of Incoming/Outgoing Links
For the process definition in Figure 2-2, the sets of outgoing and incoming links for each activity respectively are

$T^S(1) = \{(1, 3)\}$
$T^S(2) = \{(2, 1), (2, 3), (2, 4)\}$
$T^S(3) = \{(3, 2), (3, 4)\}$
$T^S(4) = \phi$
${}^PT(1) = \{(2, 1)\}$
${}^PT(2) = \{(3, 2)\}$
${}^PT(3) = \{(1, 3), (2, 3)\}$
${}^PT(4) = \{(2, 4), (3, 4)\}$

2.2 Start Activity

Start activity s, $s\in$ Sources(T), is a specified activity where a process instance can be created in the Espresso WfMS. 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.

Sources(T) = {2}

If ${}^{P}T(i) = \phi$ (i.e. 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 must be specified as a start activity of a process definition. For example, in Figure 2-3, both activity 3 and activity 10

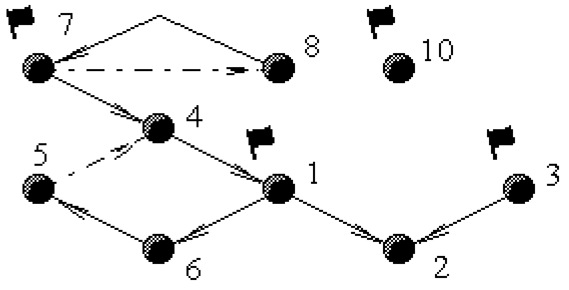


Figure 2-3. Start Activity

  have no predecessor. Therefore, they are 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 Section 4.3), such as activity 7 or activity 8 in Figure 2-3, must 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 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. Therefore, such a process definition is not valid.

Assumption 2-3. Start Activities
A process definition must have at least one start activity, i.e. Sources(T) $\neq \phi$.

PM verifies Assumption 2-3 (see the algorithm for Determining Potential Start Activities in Chapter 4), when preparing to implement a process definition (i.e. saving it as an executable version).

2.3 Algorithms for Keeping a Process Definition

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

Hypothesis

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

Principle

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

Procedures

1. When creating a new process definition:

```
$T \leftarrow \phi$
$L \leftarrow \phi$
Sources($T$) $\leftarrow \phi$
```

2. When creating activity i in a process definition:

```
$T \leftarrow T \cup \{i\}$
$T^S(i) \leftarrow \phi$
${}^PT(i) \leftarrow \phi$
```

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

```
$L \leftarrow L \cup \{(j, k)\}$
$T^S(j) \leftarrow T^S(j) \cup \{(j, k)\}$
${}^PT(k) \leftarrow {}^PT(k) \cup \{(j, k)\}$
```

4. When removing link (j, k) from a process definition:

```
$L \leftarrow L - \{(j, k)\}$
```

$T^S(j) \leftarrow T^S(j) - \{(j, k)\}$
$^PT(k) \leftarrow {}^PT(k) - \{(j, k)\}$

5. When <u>removing activity *i* from a process definition</u>:

   $T \leftarrow T - \{i\}$
   Sources(T) $\leftarrow$ Sources(T) $- \{i\}$
   call <u>removing link (i, n) from a process definition</u>, $\forall(i, n) \in T^S(i)$
   call <u>removing link (p, i) from a process definition</u>, $\forall(p, i) \in {}^PT(i)$

6. When <u>specifying activity *i* as a start activity of a process definition</u>:

   Sources(T) $\leftarrow$ Sources(T) $\cup \{i\}$

7. When <u>specifying that activity *i* is no more a start activity of a process definition</u>:

   Sources(T) $\leftarrow$ Sources(T) $- \{i\}$

2.4 Conclusion

The network of a process definition (i.e. the sets of activities, links and start activities) is determined by the process designer. The start activities, where a process instance can be created, are indicated by the designer. The end activities, where a process instance will be terminated, however, are not specified by the designer, but are determined according to the set of outgoing links of an activity (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 $^PT(i)$, $i \in T$.

Figure 2-4 illustrates the relationships between them.

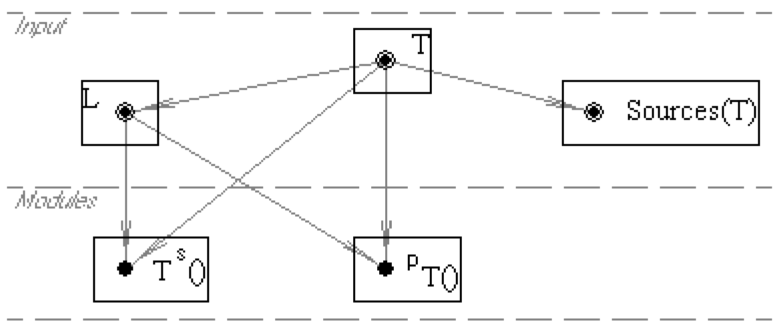


Figure 2-4. Keeping a Process Definition

3 ROUTING OPTIONS

Sequential as well as parallel routing occurs within a process instance. The following definitions are 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 the routing option of each link in a process definition together determine sequential and/or parallel routings 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 (i.e. the work is allocated to their worklists), they can execute 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 the "Multiple Choice" outgoing links of the origin activity (i.e. choose from set $\{(j, n) \mid \forall (j, n) \in T^S(j)$ with that (j, n) is a "Multiple Choice" link$\}$). Along each selected link, the work will be routed to the destination of the link.

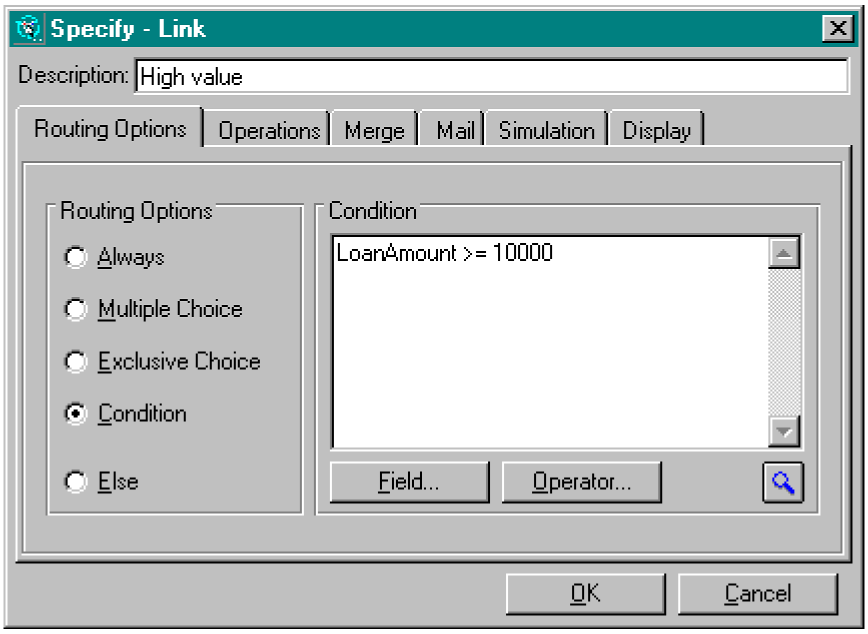
If multiple links are selected, multiple activity instances will be created after eliminating the instance of activity j.

- *Exclusive Choice*: the workflow participant who completes the origin activity of the link (activity j) should choose one and only one of the "Exclusive Choice" outgoing links of the origin activity (i.e. choose one from set $\{(j, n) \mid \forall(j, n) \in T^S(j)$ with that (j, n) 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 the given condition is met. The condition is described by a logic formula that can be evaluated by the Espresso workflow engine.
- *Else*: the work will be routed to the destination of the link (activity k), if the work cannot 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 the process instance do not terminate at the activity. It makes sense when other outgoing links of the activity are merely "Multiple Choice" or "Condition" links. That is, if $\exists(i, k) \in T^S(i)$ for which (i, k) is an "Else" link, $\forall(i, n) \in T^S(i) - \{(i, k)\}$, (i, n) is either a "Multiple Choice" or a "Condition" link. In case there are multiple "Else" outgoing links of an activity, the Espresso workflow engine uses just one of them. PM can warn such a design error.

"Multiple Choice" and "Exclusive Choice" links must be chosen at run-time by people 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 to be IT resources (see the algorithm for Getting Invalid IT Resource Activities in Section 3.2).

Routing option of a link is defined through the dialogue box shown in Figure



Description: High value

Routing Options: Always [✓ Condition] Multiple Choice Exclusive Choice Else

Condition: LoanAmount >= 10000

Figure 3-1. Routing Options

3-1. For “Multiple Choice” and “Exclusive Choice” links, descriptions of the links must be given so that people can choose routing links according to the descriptions. For a “Condition” link, a condition formula evaluated at run-time to logic value TRUE or FALSE must be given. The result of the formula tells whether to route a work along the link or not.

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 or not is made either by the workflow participant who completes it, or automatically by the workflow engine of the Espresso WfMS.

In the examples of process maps in this work, the routing option of a link can be recognized either by 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

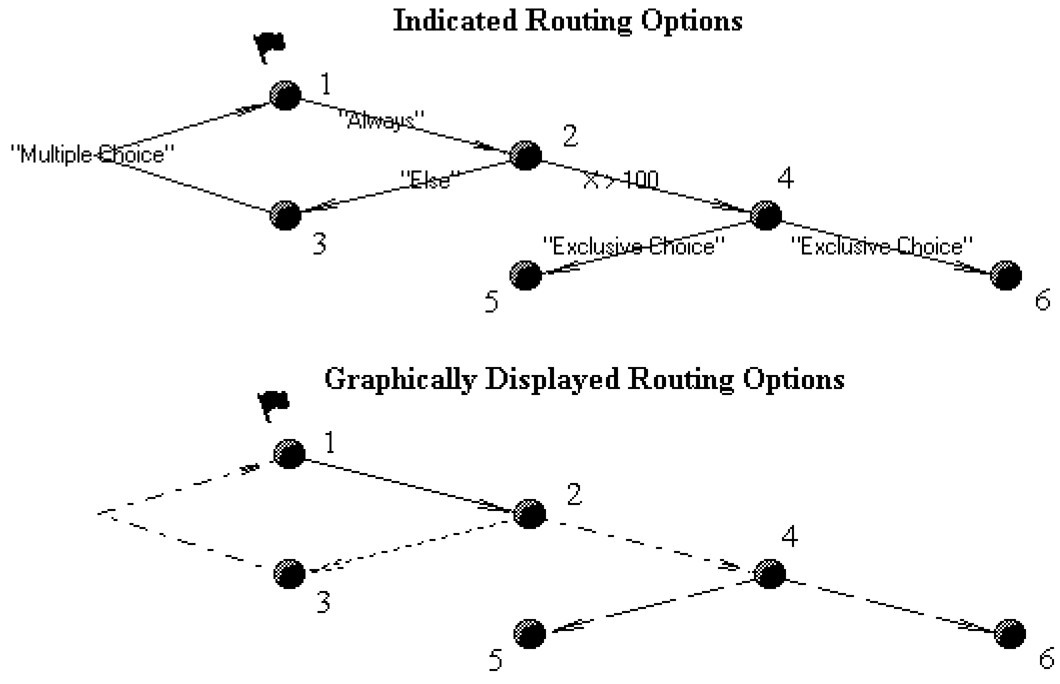


Figure 3-2. Display of Routing Options

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, if necessary.

3.2 Algorithm for <u>Getting Invalid IT Resource Activities</u>

This algorithm is used for verifying a process definition to ensure that no choice link is an outgoing link of the activity completed by an IT Resource.

<u>Hypothesis</u>
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.

<u>Principle</u>
"Multiple Choice" and "Exclusive Choice" links must be chosen at run-time by people. 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 set RestLinks keeps not treated outgoing links of the current activity.

<u>Procedure</u>

```
Step 1:  InvalidActivities ← $\phi$;
Step 2:  RestActivities ← $T$;
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 ← $T^S(i)$;
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 ← InvalidActivities $\cup$ $\{i\}$; go to Step 3;
Step 12: stop (return InvalidActivities).
```

3.3 End Activity

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

A *structural end activity* of a process definition is the activity where a work associated with a process instance of the process definition is possible to be terminated. 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.

In other words, an activity is not a structural end activity if it has an outgoing link with the routing option of "Always", "Exclusive Choice" or "Else".

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, a work associated with a process instance cannot 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 evaluated to TRUE.

<u>Example 3-1. End Activity</u>

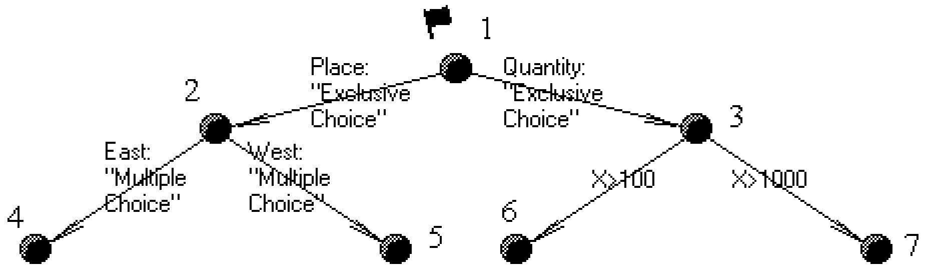


Figure 3-3. Structural End Activities 2, 4, 5, 6 and 7

In Figure 3-3, activities 4, 5, 6 and 7 are structural end activities that must belong to set Sinks(T), because they have no successor, i.e.

$$T^S(4) = T^S(5) = T^S(6) = T^S(7) = \phi$$

Activity 3 is a structural end activity that can be specified as a non-end activity of the process definition, since all of 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 will 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, since all of 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

In the Espresso WfMS, the flexible activity execution threads of process instances can be realized via the routing option specification. A link connecting two activities can be one of five routing options of “Always”, “Multiple Choice”, “Exclusive Choice”, “Condition” and “Else”. The routing option makes it possible to route a work along a link under certain decisions or conditions associated with a business process.

Whether to route a work along a “Multiple Choice” or “Exclusive Choice” link or not is decided by the workflow participants who complete the origin activity. Therefore, it will be verified that such kinds of routing options are not included in the outgoing links of an activity executed 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 how invalid IT resource activities and end activities are determined from process definition data.

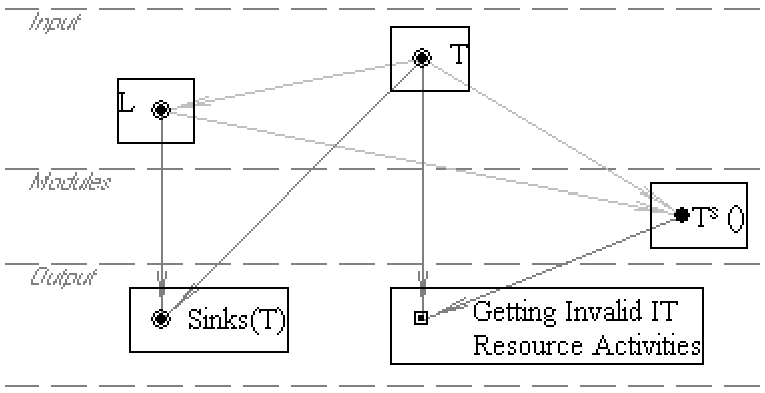


Figure 3-4. Getting End and Invalid IT Resource Activities

4 PATH AND CYCLE

Every activity of a process definition must be reachable from a start activity via a path. Infinite cycle in the network of the process definition is not allowed.

4.1 Paths

In a process definition, *path*s 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

$$(i, (i, q_1), q_1, (q_1, q_2), q_2, ..., q_n, (q_n, k), k)$$

or simply by a sequence of activities as

$$(i, q_1, q_2, ..., q_n, k)$$

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

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 an instance of activity k.

Theorem 4-1. Path Transitivity
If $i \rightarrow k$ and $k \rightarrow j$ exist, then $i \rightarrow j$ exists.

<u>Proof</u>

1° because $\exists i \rightarrow k$, so
  $\exists q_1, q_2, ..., q_n, i, k \in T$ and $(i, q_1), (q_1, q_2), ..., (q_n, k) \in L$ with
    $(i, (i, q_1), q_1, (q_1, q_2), q_2, ..., q_n, (q_n, k), k)$

2° because $\exists k \rightarrow j$, so
  $\exists p_1, p_2, ..., p_m, k, j \in T$ and $(k, p_1), (p_1, p_2), ..., (p_m, j) \in L$ with
    $(k, (k, p_1), p_1, (p_1, p_2), p_2, ..., p_m, (p_m, j), j)$

3° from the results of steps 1 and 2, we get

$(i, (i, q_1), q_1, (q_1, q_2), q_2, ..., q_n, (q_n, k), k, (k, p_1), p_1, (p_1, p_2), p_2, ..., p_m, (p_m, j), j)$

That is, a path of $i \rightarrow j$ exists.

End of proof

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

Example 4-1. Path

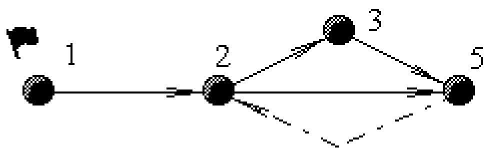


Figure 4-1. Path

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→2, 2→3, 2→5, 3→5 and 5→2

Now according to the second path definition, from paths as well as links

1→2, (2, 3)
1→2, (2, 5)
2→3, (3, 5)
2→5, (5, 2)
3→5, (5, 2)
5→2, (2, 3) and
5→2, (2, 5)

respectively, the following paths exist:

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

Again according to the second definition, from paths as well as links

(1, 2, 3), (3, 5)
(1, 2, 5), (5, 2)
(2, 3, 5), (5, 2)
(2, 5, 2), (2, 3)
(2, 5, 2), (2, 5)
(3, 5, 2), (2, 3)
(3, 5, 2), (2, 5)
(5, 2, 3), (3, 5) and
(5, 2, 5), (5, 2)

respectively, we know that there exist paths

$1 \rightarrow 5$ (i.e. (1, 2, 3, 5))
$1 \rightarrow 2$ (i.e. (1, 2, 5, 2))
$2 \rightarrow 2$ (i.e. (2, 3, 5, 2))
$2 \rightarrow 3$ (i.e. (2, 5, 2, 3))
$2 \rightarrow 5$ (i.e. (2, 5, 2, 5))
$3 \rightarrow 3$ (i.e. (3, 5, 2, 3))
$3 \rightarrow 5$ (i.e. (3, 5, 2, 5))
$5 \rightarrow 5$ (i.e. (5, 2, 3, 5)) and
$5 \rightarrow 2$ (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 *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

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

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

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

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

Elementary path of 2→3 is (2, 3), and hence

Activities(2→3) = {2, 3}

An elementary path of $i \to k$ gives the shortest way for a work to flow from activity i to activity k over activities within set Activities($i \to k$).

Theorem 4-2. Elementary Path
If a path of $i \to k$ exists, an elementary path of $i \to k$ exists too.

Proof

1° Because $\exists i \to k$, so
$\exists q_1, q_2, ..., q_n, i, k \in T$ and $(i, q_1), (q_1, q_2), ..., (q_n, k) \in L$ with
$(i, (i, q_1), q_1, (q_1, q_2), q_2, ..., q_n, (q_n, k), k)$

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

3° suppose that only $i = q_j, j = 1, 2, \dots$, or n, then we have
$(q_j, (q_{j+1}, q_{j+2}), q_{j+2}, ..., q_n, (q_n, k), k)$ or
$(i, (q_{j+1}, q_{j+2}), q_{j+2}, ..., q_n, (q_n, k), k)$
That is, an elementary path of $i \to k$ exists;

4° suppose that only $k = q_j, j = 1, 2, \dots$, or n, then exists
$(i, (i, q_1), q_1, ..., q_{j-1}, (q_{j-1}, q_j), q_j)$ or
$(i, (i, q_1), q_1, ..., q_{j-1}, (q_{j-1}, q_j), k)$
That is, an elementary path of $i \to k$ exists;

5° suppose that only $q_j = q_f, j, f = 1, 2, \dots$, or n, and $j < f$, path
$(i, (i, q_1), q_1, (q_1, q_2), q_2, ..., q_n, (q_n, k), k)$
can then be denoted as
$(i, (i, \mathrm{q}_1), q_1, ..., (q_{j-1}, q_j), q_j, (q_j, q_{j+1}), ..., (q_{f-1}, q_f), q_f, (q_f, q_{f+1}), ..., q_n, (q_n, k), k)$
because $q_j = q_f$, so exists path
$(i, (i, q_1), q_1, ..., (q_{j-1}, q_j), q_j, (q_f, q_{f+1}), ..., q_n, (q_n, k), k)$
that is, an elementary path of $i \to k$.

From the results of steps 3, 4 and 5 we know that any non-elementary path of $i \to k$ can be transformed to an elementary path of $i \to k$.

End of Proof

From this theorem, it can be deducted that if no elementary path of $i \to k$ exists, no path of $i \to k$ exists either.

4.1.2 Algorithm for Finding an Elementary Path*

This algorithm is, according to the set of outgoing links of each activity, to find an elementary path between two activities within a process definition without passing over one of the given activities.

This algorithm is not used anymore in verification and simulation of a process definition, because performance problems arise if many paths between two different activities are needed. But it could be used for development of a new feature that requires just one path between two activities.

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 $i \neq k$, and $U \subset T$. This procedure will return a certain path of $i \rightarrow k$, if there exists an elementary path from activity *i* to activity *k* and the path does not include any activities in set *U*; otherwise return empty.

To determine whether such a path exists, a self-called sub procedure with the first 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 PathStack(*j*), *j* = 1, 2, ..., StackPointer. When the sub procedure is called by the main procedure, StackPointer is assigned with 0. Thus, activity *i*, the beginning activity of the path, is kept in PathStack(1).

Temporary set RestLinks is used in the sub procedure for keeping the not treated links. Variable CurPath keeps the path.

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

```
Step 1:  if $i \in U$ or $k \in U$, stop (return empty);
Step 2:  StackPointer $\leftarrow$ 0;
Step 3:  CurPath $\leftarrow$ call the sub procedure with parameters $i$, $k$ and $U$;
Step 4:  stop (return CurPath).
```

Sub Procedure (*q*, *k*, *U*)

```
Step 1:  StackPointer $\leftarrow$ StackPointer + 1, PathStack(StackPointer) $\leftarrow$ $q$ (put $q$
         in stack);
Step 2:  if $q = k$ (paths $i \rightarrow k$ exist), stop (return activity sequence of
         PathStack(1), PathStack(2), …, PathStack(StackPointer));
Step 3:  RestLinks $\leftarrow$ $T^S(q)$;
Step 4:  if RestLinks = $\phi$, go to Step 10;
Step 5:  remove an element, say link $(q, n)$, from set 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 $n$ = PathStack($j$) with $j \in$ [1, StackPointer] (activity $n$ is on current
         searching path), go to Step 4;
Step 8:  CurPath ← call the sub procedure self with parameters $n$, $k$ and $U$; If
         CurPath is not empty (paths $i \to q \to n \to k$ exist), stop (return CurPath);
Step 9:  go to Step 4;
Step 10: (paths $q \to k$ don't exist) StackPointer ← 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 $i \to k$ exists.

Example 4-2. Set of Reachable Activities

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

$1 \to 2$, $2 \to 3$, $2 \to 5$, $3 \to 5$, $5 \to 2$
$1 \to 3$, $1 \to 5$, $2 \to 5$, $2 \to 2$, $3 \to 2$, $5 \to 3$, $5 \to 5$
$1 \to 5$, $1 \to 2$, $2 \to 2$, $2 \to 3$, $2 \to 5$, $3 \to 3$, $3 \to 5$, $5 \to 5$, $5 \to 2$, etc.

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
$i \to k$ and $p \to i$
According to Theorem 4-1, exists a path of
$p \to k$
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. Therefore, it makes no sense to define such an activity in a process definition. PM can verify the following assumption.

Assumption 4-1. Activity Reachability
Every activity 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 the network of 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 $Paths(i \rightarrow k)$.

According to the Theorem 4-2, if a path of $i \rightarrow k$ exists, $\text{Paths}(i \rightarrow k) \neq \phi$. 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)

Thus,

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

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

$\forall i \rightarrow k \in \text{Paths}(i \rightarrow k)$ 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 $Criticals(i \rightarrow k)$.

Example 4-3. Set of Critical Activities
For the process definition shown in Figure 4-1,

$\text{Paths}(1 \rightarrow 1) = \phi$,	$\text{Criticals}(1 \rightarrow 1) = \phi$
$\text{Paths}(1 \rightarrow 2) = \{(1, 2)\}$,	$\text{Criticals}(1 \rightarrow 2) = \phi$
$\text{Paths}(1 \rightarrow 3) = \{(1, 2, 3)\}$,	$\text{Criticals}(1 \rightarrow 3) = \{2\}$
$\text{Paths}(1 \rightarrow 5) = \{(1, 2, 3, 5), (1, 2, 5)\}$,	$\text{Criticals}(1 \rightarrow 5) = \{2\}$

Paths($2\rightarrow1$) = ϕ,	Criticals($2\rightarrow1$) = ϕ
Paths($2\rightarrow2$) = {(2, 5, 2), (2, 3, 5, 2)},	Criticals($2\rightarrow2$) = {5}
Paths($2\rightarrow3$) = {(2, 3)},	Criticals($2\rightarrow3$) = ϕ
Paths($2\rightarrow5$) = {(2, 5), (2, 3, 5)},	Criticals($2\rightarrow5$) = ϕ
Paths($3\rightarrow1$) = ϕ,	Criticals($3\rightarrow1$) = ϕ
Paths($3\rightarrow2$) = {(3, 5, 2)},	Criticals($3\rightarrow2$) = {5}
Paths($3\rightarrow3$) = {(3, 5, 2, 3)},	Criticals($3\rightarrow3$) = {2, 5}
Paths($3\rightarrow5$) = {(3, 5)},	Criticals($3\rightarrow5$) = ϕ
Paths($5\rightarrow1$) = ϕ,	Criticals($5\rightarrow1$) = ϕ
Paths($5\rightarrow2$) = {(5, 2)},	Criticals($5\rightarrow2$) = ϕ
Paths($5\rightarrow3$) = {(5, 2, 3)},	Criticals($5\rightarrow3$) = {2}
Paths($5\rightarrow5$) = {(5, 2, 5), (5, 2, 3, 5)},	Criticals($5\rightarrow5$) = {2}

That is, activity 2 is a critical activity of paths $1\rightarrow3$, $1\rightarrow5$, $3\rightarrow3$, $5\rightarrow3$ and $5\rightarrow5$; Activity 5 is a critical activity of paths $2\rightarrow2$, $3\rightarrow2$ and $3\rightarrow3$.

4.2.3 Algorithm for Determining Reachability and Criticality

This 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. Reaches(i), $\forall i\in T$, denotes a set of activities reachable from activity i. Criticals($i\rightarrow k$), $\forall i, k\in T$, stands for the set of critical activities of paths $i\rightarrow k$.

Principle

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

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

Procedure

```
Step 1:  Reaches($i$) $\leftarrow \phi$ and Previous($i$) $\leftarrow \phi$, $\forall i\in T$;
Step 2:  Criticals($i\rightarrow k$) $\leftarrow T$, $\forall i, k\in T$;
Step 3:  RestLinks $\leftarrow L$;
Step 4:  if RestLinks = $\phi$, go to Step 13;
Step 5:  remove an element, say link ($i$, $k$), from RestLinks;
Step 6:  $\forall p\in$ Previous($i$), let
             Reaches($p$) $\leftarrow$ Reaches($p$) $\cup$ {$k$} $\cup$ Reaches($k$)
             Criticals($p\rightarrow k$) $\leftarrow$ Criticals($p\rightarrow k$) $\cap$ (Criticals($p\rightarrow i$) $\cup$ {$i$})
```

```
          Criticals($p \rightarrow s$) $\leftarrow$ Criticals($p \rightarrow s$) $\cap$ (Criticals($p \rightarrow i$) $\cup$ $\{i\}$ $\cup$ $\{k\}$ $\cup$
                    Criticals($k \rightarrow s$)), $\forall s \in$ Reaches($k$)
Step 7:  $\forall n \in$ Reaches($k$), let
          Previous($n$) $\leftarrow$ Previous($n$) $\cup$ $\{i\}$ $\cup$ Previous($i$)
Step 8:  Reaches($i$) $\leftarrow$ Reaches($i$) $\cup$ $\{k\}$ $\cup$ Reaches($k$);
Step 9:  Previous($k$) $\leftarrow$ Previous($k$) $\cup$ $\{i\}$ $\cup$ Previous($i$);
Step 10: Criticals($i \rightarrow k$) $\leftarrow \phi$ (link ($i$, $k$) makes no critical activity existing on
         paths of $i \rightarrow k$);
Step 11: $\forall s \in$ Reaches($k$), let
          Criticals($i \rightarrow s$) $\leftarrow$ Criticals($i \rightarrow s$) $\cap$ ($\{k\}$ $\cup$ Criticals($k \rightarrow s$))
Step 12: go to Step 4;
Step 13: Criticals($i \rightarrow k$) $\leftarrow \phi$, $\forall i, k \in T$ with $k \notin$ Reaches($i$);
Step 14: stop (Reaches($i$), Criticals($i \rightarrow k$), $\forall i \in T$ and $\forall k \in$ Reaches($i$), have been
         determined).
```

Example 4-4. Determine Reachable/Critical Activities

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

i	1	2	3	5
Reaches(i)	ϕ	ϕ	ϕ	ϕ
Previous(i)	ϕ	ϕ	ϕ	ϕ

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}
$k = 2$	{1, 2, 3, 5}	{1, 2, 3, 5}	{1, 2, 3, 5}	{1, 2, 3, 5}
$k = 3$	{1, 2, 3, 5}	{1, 2, 3, 5}	{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}

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

i	1	2	3	5
Reaches(i)	{**2**}	ϕ	ϕ	ϕ
Previous(i)	ϕ	{**1**}	ϕ	ϕ

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}
$k = 2$	$\boldsymbol{\phi}$	{1, 2, 3, 5}	{1, 2, 3, 5}	{1, 2, 3, 5}
$k = 3$	{1, 2, 3, 5}	{1, 2, 3, 5}	{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}

RestLinks = {(2, 3), (2, 5), (3, 5), (5, 2)};

After treating link (2, 3), and going back to Step 4, the values become

i	1	2	3	5
Reaches(*i*)	{2, **3**}	{**3**}	ϕ	ϕ
Previous(*i*)	ϕ	{1}	{**2, 1**}	ϕ

Criticals(*i*→*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	ϕ	{1, 2, 3, 5}	{1, 2, 3, 5}	{1, 2, 3, 5}
k = 3	{**2**}	**ϕ**	{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}

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

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

i	1	2	3	5
Reaches(*i*)	{2, 3, **5**}	{3, **5**}	ϕ	ϕ
Previous(*i*)	ϕ	{1}	{2, 1}	{**2, 1**}

Criticals(*i*→*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	ϕ	{1, 2, 3, 5}	{1, 2, 3, 5}	{1, 2, 3, 5}
k = 3	{2}	ϕ	{1, 2, 3, 5}	{1, 2, 3, 5}
k = 5	{**2**}	**ϕ**	{1, 2, 3, 5}	{1, 2, 3, 5}

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

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

i	1	2	3	5
Reaches(*i*)	{2, 3, 5}	{3, 5}	{**5**}	ϕ
Previous(*i*)	ϕ	{1}	{2, 1}	{2, 1, **3**}

Criticals(*i*→*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	ϕ	{1, 2, 3, 5}	{1, 2, 3, 5}	{1, 2, 3, 5}
k = 3	{2}	ϕ	{1, 2, 3, 5}	{1, 2, 3, 5}
k = 5	{2}	**ϕ**	**ϕ**	{1, 2, 3, 5}

RestLinks = {(5, 2)}.

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

i	1	2	3	5
Reaches(*i*)	{2, 3, 5}	{3, 5, **2**}	{5, **2, 3**}	{**2, 3, 5**}
Previous(*i*)	ϕ	{1, **5, 2, 3**}	{2, 1, **5, 3**}	{2, 1, 3, **5**}

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	**ϕ**	**{5}**	**{5}**	**ϕ**
k = 3	**{2}**	**ϕ**	**{5, 2}**	**{2}**
k = 5	**{2}**	**ϕ**	**ϕ**	**{2}**

RestLinks = ϕ;

Now because RestLinks = ϕ, go to Step 13, where Criticals(1→1), Criticals(2→1), Criticals(3→1) and Criticals(5→1) are set to ϕ, for 1∉Reaches(1), 1∉Reaches(2), 1∉Reaches(3), and 1∉Reaches(4). Eventually reachable and critical values are

i	1	2	3	5
Reaches(*i*)	{2, 3, 5}	{3, 5, 2}	{5, 2, 3}	{2, 3, 5}

Criticals($i \rightarrow k$)	*i* = 1	*i* = 2	*i* = 3	*i* = 5
k = 1	ϕ	ϕ	ϕ	ϕ
k = 2	ϕ	{5}	{5}	ϕ
k = 3	{2}	ϕ	{5, 2}	{2}
k = 5	{2}	ϕ	ϕ	{2}

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

4.3 Cycles

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

Example 4-5. Cycle

In Figure 4-1 of Example 4-1, since (5, 2), (2, 3), (2, 5), (3, 5)∈L, each of the following paths obtained in Example 4-3 builds up an element cycle with a corresponding link:

(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,

cycle (2, 5) ≡ cycle (5, 2)

Similarly,

cycle (2, 3, 5) ≡ cycle (3, 5, 2) ≡ cycle (5, 2, 3)

If a cycle is not reachable from any activity outside the cycle, the cycle is a *structural start cycle*. At least one of the activities on a structural start cycle must be a start activity; otherwise the activities on the cycle are not reachable from a start activity of a process definition.

A cycle is an *infinite cycle*, if a work involved in 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, so that any process instance created in a WfMS can eventually terminate 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. Therefore, 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 or not. $T^S(i)$ 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 $\exists s \in$ Sinks(T) with activity i has a path to activity s. In the meantime, ToEnd(n), $\forall(i, n) \in T^S(i)$, will also be determined. Variable NewlyDetermined, keeping the number of newly determined activities, will be updated 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 are 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 or not cannot 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 or not cannot 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 = $\phi$, 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 ← TotalDetermined + NewlyDetermined; go to Step
         3;
Step 8:  if TotalDetermined $\neq 0$ (some activities have been newly 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($i$), $\partial i \in T$).
```

<u>Sub Procedure (*i*)</u>

```
Step 1:  if $T^S(i) = \phi$ (activity $i$ has no outgoing link, so it cannot be specified as
         a non-end activity),
             1° let
                 ToEnd($i$) ← TRUE
                 NewlyDetermined ← NewlyDetermined + 1
             2° stop;
Step 2:  StackPointer ← StackPointer + 1; PathStack(StackPointer) ← $i$;
Step 3:  ALs ← 0; XLs ← 0; ELs ← 0;
Step 4:  ALEnds ← 0; XLEnds ← 0; TotalLEnds ← 0;
Step 5:  ALNils ← 0; XLNils ← 0; TotalLNils ← 0;
Step 6:  RestLinks ← $T^S(i)$;
Step 7:  remove an element, say link $(i, n)$, from set RestLinks;
Step 8:  if $(i, n)$ is an "Always" link, then ALs ← ALs + 1;
         if $(i, n)$ is an "Exclusive Choice" link, then XLs ← XLs + 1;
         if $(i, n)$ is an "Else" link, then ELs ← ELs + 1;
Step 9:  if ToEnd($n$) is not empty (activity $n$ as well as its successors have been
         dealt with), go to Step 13;
Step 10: if $n$ = PathStack($j$) with $j \in [1$, StackPointer] (activity $n$ is on the
         searching path), go to Step 12;
Step 11: call the sub procedure self with parameter $n$;
Step 12: if ToEnd($n$) is empty (whether activity $n$ has a path to an end activity
         or not cannot be determined),
             1° let TotalLNils ← TotalLNils + 1;
             2° if $(i, n)$ is an "Always" link, then
                     ALNils ← ALNils + 1
             3° if $(i, n)$ is an "Exclusive Choice" link, then
                     XLNils ← XLNils + 1
             4° go to Step 14;
Step 13: if ToEnd($n$) = TRUE,
             1° let TotalLEnds ← TotalLEnds + 1;
             2° if $(i, n)$ is an "Always" link, then
                     ALEnds ← ALEnds + 1
             3° if $(i, n)$ is an "Exclusive Choice" link, then
                     XLEnds ← XLEnds + 1
Step 14: if RestLinks $\neq \phi$, go to Step 7;
Step 15: if ALs = 0, XLs = 0 and ELs =0, but $i \notin$ Sinks($T$) (activity $i$ is specified
         as a non-end activity), let ELs ← ELs + 1;
```

```
Step 16: if ALs = ALEnds, go to Step 18;
Step 17: (at least one "Always" link has no path to an end activity) if ALNils >
         0 (ToEnd(i) cannot be determined), go to Step 24; otherwise (a work
         at activity i can never be terminated) let ToEnd(i) ← FALSE and go to
         Step 23;
Step 18: if XLs = 0 or XLEnds > 0, go to Step 20;
Step 19: (at least along one of the "Exclusive Choice" links the work flows
         further, but none of the destinations of the links have paths to an end
         activity) if XLNils > 0 (ToEnd(i) cannot be determined), go to Step
         24; otherwise let ToEnd(i) ← 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 will flow further, but no successor
         of activity i has paths to an end activity) if TotalLNils > 0, go to Step
         24; otherwise let ToEnd(i) ← FALSE and go to Step 23;
Step 22: ToEnd(i) ← TRUE;
Step 23: NewlyDetermined ← NewlyDetermined + 1;
Step 24: StackPointer ← 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 the links in set U. Current 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 found path combines a cycle with link (k, i) and so 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∈ T, will be
        determined);
Step 2: Cycles ← $\phi$;
Step 3:  RestLinks ← $L$;
Step 4:  if |RestLinks| ≤ 1 (links in set RestLinks cannot combine any cycle),
         stop (return Cycles);
Step 5:  remove an element, say link ($k$, $i$), from set RestLinks;
Step 6:  StackPointer ← 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 ← StackPointer + 1, PathStack(StackPointer) ← $i$;
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 ← $T^S(i)$;
Step 4:  if RestLinks = $\phi$, 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 $k \notin$ Reaches($n$) (activity $n$ has no path to activity $k$), go to Step 4;
Step 8:  if $n$ = PathStack($j$) with $j \in [1$, 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 ← StackPointer – 1; stop.
```

4.3.3 Algorithm for Getting Infinite Elementary Cycles

This algorithm is used for verification of a process definition to ensure that there is no infinite cycle in 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 or not.

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 Determining Activity Reachable to End (ToEnd($i$), $i \in T$, will be
        updated);
Step 2: RestCycles ← Getting Elementary Cycles;
Step 3: InfiniteCycles ← ϕ;
Step 4: if RestCycles = ϕ, 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($q$) = TRUE, cycle CurCycle is not infinite; otherwise add cycle
        CurCycle to set InfiniteCycles;
Step 7: go to Step 4.
```

Example 4-6. Determine Infinite Cycles

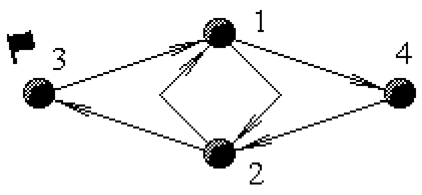
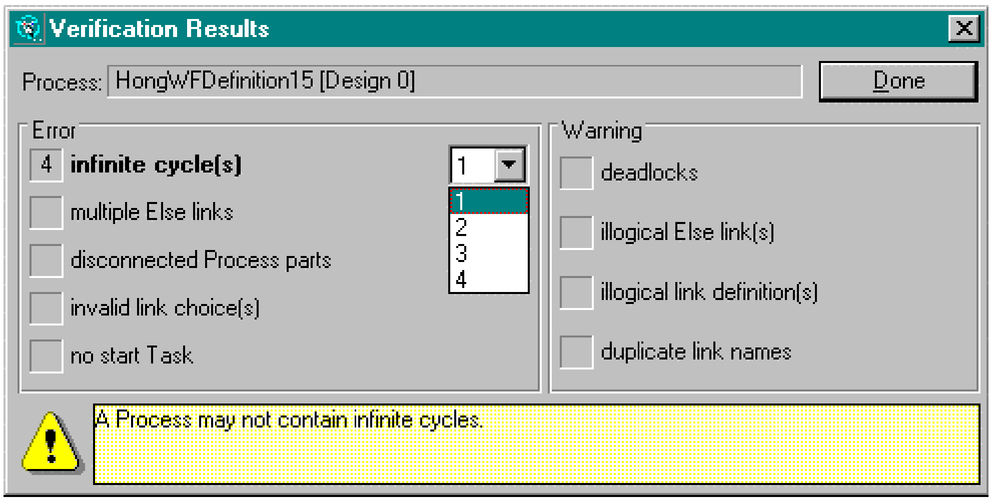


Figure 4-2. Cycle

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.


Verification Results

Process: HongWFDefinition15 [Design 0]

Error

4 infinite cycle(s)

multiple Else links

disconnected Process parts

invalid link choice(s)

no start Task

Warning

deadlocks

illogical Else link(s)

illogical link definition(s)

duplicate link names

A Process may not contain infinite cycles.

Figure 4-3. Verification Results—Infinite Cycle

When an infinite cycle is selected in the dialogue box, it will be indicated on the process map by square-marked links that make up the infinite 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) and
cycle (1, 4, 2, 3)

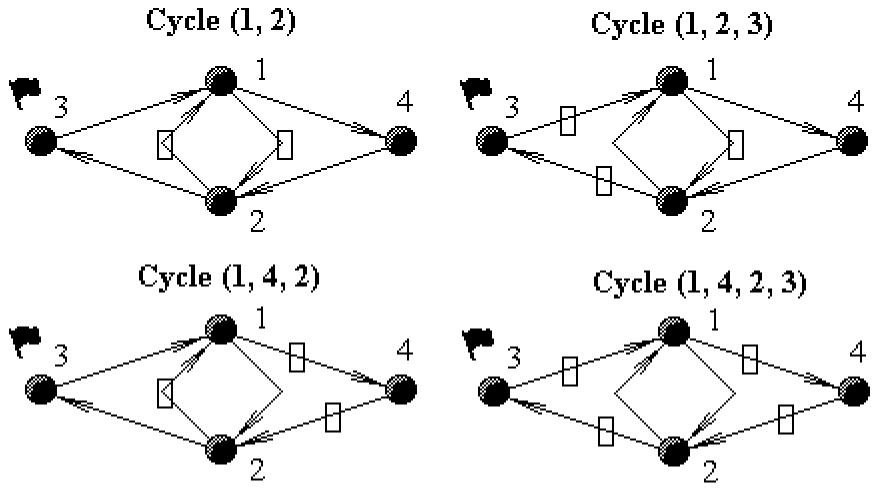


Figure 4-4. Infinite Cycles

4.4 Algorithm for Determining Potential Start Activities

This algorithm is used for verification of a process definition to ensure that every activity in a process definition is reachable from a start activity. According to the rules discussed in Section 2.2 for specifying start activities of a process definition, all the structural start activities and one of an activities involved in a structural start cycles must be specified as a start activity.

Hypothesis
T denotes the activity set of a process definition. Sources(*T*) stands for the set of start activities. ${}^{P}T(i)$, $i \in T$, 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 must be specified as start activities so that every activity in a process definition is 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. One activity on a structural start cycle is also 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 $\leftarrow \phi$;
Step 2:  $\forall i \in T$, if $^{P}T(i) = \phi$ 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 $\leftarrow \phi$;
Step 4:  RestCycles $\leftarrow$ Getting Elementary Cycles;
Step 5:  if RestCycles = $\phi$, go to Step 11;
Step 6:  remove a cycle, say $i \rightarrow k \rightarrow (i)$, 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 $\exists s \rightarrow q \rightarrow (s) \in$ CycleStarts with $i \in$ Reaches($s$) (activity $i$ is reachable
         from a cycle in set CycleStarts), go to Step 5;
Step 9:  $\forall s \rightarrow q \rightarrow (s) \in$ CycleStarts, if $s \in$ Reaches($i$) (cycle $s \rightarrow q \rightarrow (s)$ is reachable
         from activity $i$), let
             CycleStarts $\leftarrow$ CycleStarts $- \{s \rightarrow q \rightarrow (s)\}$
Step 10: CycleStarts $\leftarrow$ CycleStarts $\cup \{i \rightarrow k \rightarrow (i)\}$; go to Step 5;
Step 11: (now all the cycles in set CycleStarts are structural start cycles and one
         is not reachable from another) NotSpecified $\leftarrow$ NotSpecified $\cup \{i\}$,
         $\forall i \rightarrow k \rightarrow (i) \in$ CycleStarts;
Step 12: stop (return NotSpecified).
```

Example 4-7. Determine Potential Start Activities
For the process definition in Figure 4-5, activity 1 is specified as a start activity.

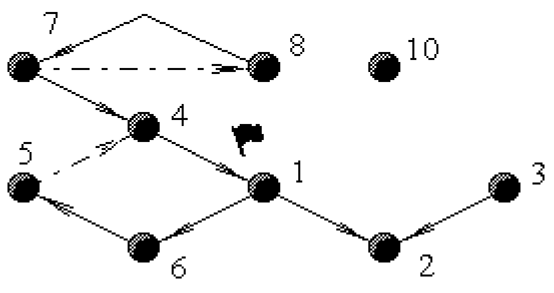
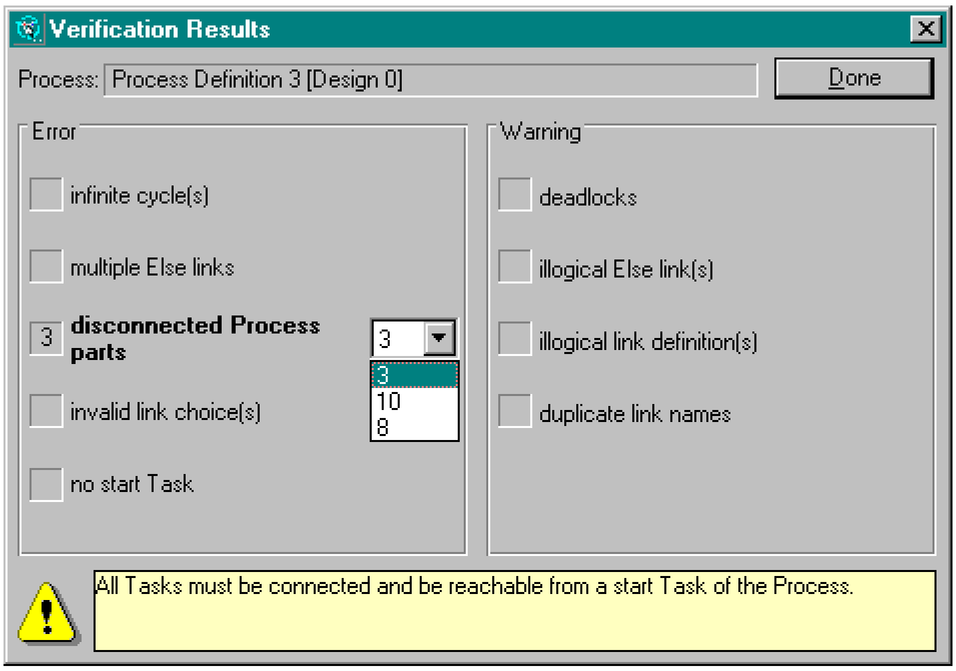


Figure 4-5. Potential Start Activities

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 on the cycle, here activity 8, is determined as a potential start activity.

The dialogue box "Verification Results" shown in Figure 4-6 informs that there are three disconnected process parts beginning with potential start 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

Process: Process Definition 3 [Design 0]

Done

Error

infinite cycle(s)

multiple Else links

3 **disconnected Process parts** 3

3
10
8

invalid link choice(s)

no start Task

Warning

deadlocks

illogical Else link(s)

illogical link definition(s)

duplicate link names

All Tasks must be connected and be reachable from a start Task of the Process.

Figure 4-6. Verification Results—Start Activity

4.5 Conclusion

Paths from activity *i* to activity *k*, denoted by *i*→*k*, are defined to explain whether activity *k* is reachable from activity *i*. That is, if a path of *i*→*k* exists, a work at activity *i* is possible to flow to activity *k*.

Every non-start activities within a process definition must be reachable from a start activity. The potential start activities are the activities that must be specified further as the start activities of a process definition.

An activity *p* is critical on paths of *i*→*k*, denoted by *p*∈Criticals(*i*→*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 Reaches(*i*). Sets Reaches(*i*) and Criticals(*i*→*k*), ∀*i*, *k*∈*T*, are determined together by an algorithmic procedure.

A cycle is a special path of *i*→*k*, for which (*k*, *i*)∈*L*. A work involved in an infinite cycle can never be terminated. Therefore, infinite cycles are not allowed to exist in a process definition. They are detected by PM through checking whether an activity on the cycle, say activity *i*, has a path to an end activity, denoted by 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. It should be mentioned that the algorithms with a self-called sub procedure have performance problem, when many infinite cycles (e.g. 400) are modeled in the network of a process definition.

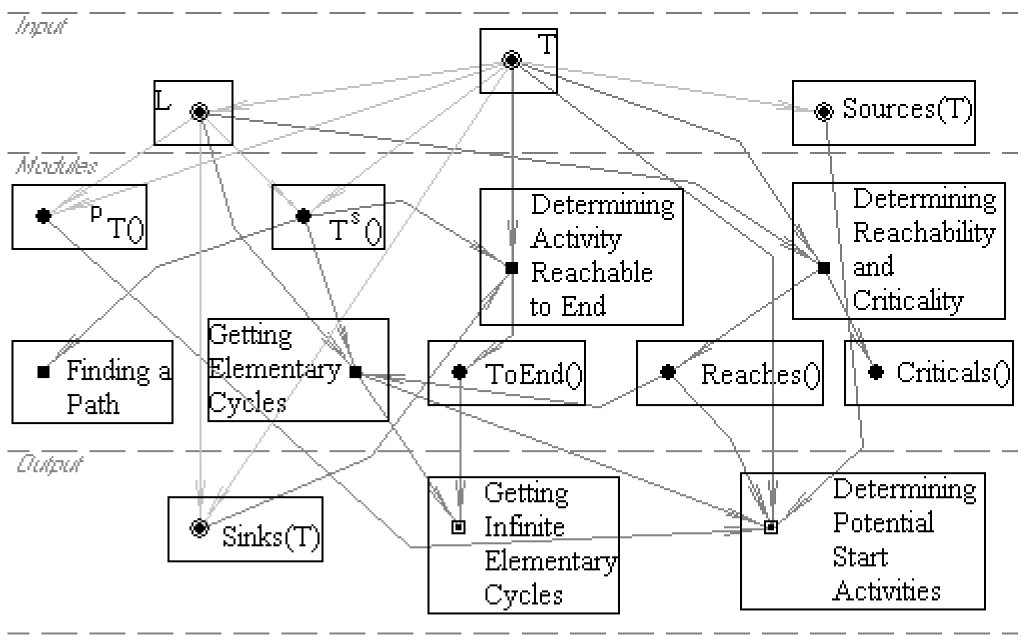

Input
T
L
Sources(T)
Modules
^{P}T()
T^s()
Determining Activity Reachable to End
Determining Reachability and Criticality
Finding a Path
Getting Elementary Cycles
ToEnd()
Reaches()
Criticals()
Output
Sinks(T)
Getting Infinite Elementary Cycles
Determining Potential Start Activities

Figure 4-7. Getting Potential Start Activities, Paths, and Infinite Cycles

5 SPLIT AND JOIN

A split activity causes parallel routing of work items associated with the same process instance. Parallel work items will be automatically joined at a join activity in the Espresso WfMS. The join activities determined by PM are used by the Espresso workflow engine to decide where to join parallel work items.

5.1 Split Activity

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

1. $|T^S(i)| > 1$;
2. $\exists(i, n) \in T^S(i)$, (i, n) is neither an "Exclusive Choice" link nor an "Else" link; and
3. when $|T^S(i)| = 2$, $\forall(i, n) \in T^S(i)$, (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 into multiple concurrent execution threads and parallel activity instances of a process instance consequently exist in the Espresso WfMS.

The set of split activities of a process definition is denoted by *Splits*(T). Splits(T) $\subset T$. If Splits(T) $\neq \phi$, it is possible that multiple activity instances are associated with a common process instance.

Example 5-1. Split Activity

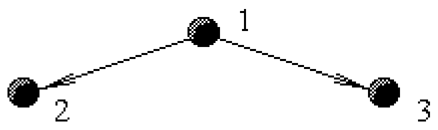


Figure 5-1. Split Activity

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

1. $T^S(1) = \{(1, 2), (1, 3)\}$ (i.e. $|T^S(1)| = 2$); 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, 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. Not a Split Activity—Exclusive Links

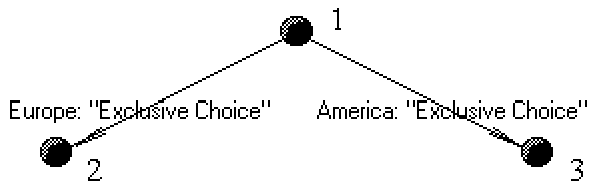


Figure 5-2. Not a Split Activity—Exclusive Links

In Figure 5-2, the two links named with "Europe" and "America" respectively are both "Exclusive Choice" 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, only one of the links (1, 2) and (1, 3) can be selected to let the work be routed along it.

Example 5-3. Not a Split Activity—Else Link

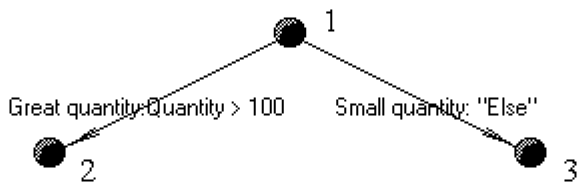


Figure 5-3. Not a Split Activity—Else Link

In Figure 5-3, link "Great quantity" is a "Condition" link and link "Small quantity" is an "Else" link.

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. $T^S(i)$, $\forall i \in T$, represents the set of outgoing links of activity i. Splits(T) stands for the set of split activities of the process definition.

Principle

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

Procedure

```
Step 1:  Splits($T$) $\leftarrow \phi$;
Step 2:  RestActivities $\leftarrow T$;
Step 3:  remove an element, say activity $i$, from set RestActivities;
Step 4:  if $|T^S(i)| < 2$ (activity $i$ has no or only one outgoing link), go to Step 12;
Step 5:  if $|T^S(i)| > 2$, go to Step 7;
Step 6:  (activity $i$ has just two outgoing links) get the two links from $T^S(i)$, say
         $(i, n)$ and $(i, k)$; if $(i, n)$ or $(i, k)$ is an "Else" link, go to Step 12;
Step 7:  RestLinks $\leftarrow T^S(i)$;
Step 8:  remove an element, say $(i, n)$, from set RestLinks;
Step 9:  if $(i, n)$ is neither an "Exclusive Choice" link nor an "Else" link, go to
         Step 11;
Step 10: if RestLinks $= \phi$ (all outgoing links of activity $i$ are "Exclusive
         Choice" or "Else" links), go to Step 12; otherwise go to Step 8;
Step 11: Splits($T$) $\leftarrow$ Splits($T$) $\cup \{i\}$;
Step 12: if RestActivities $\neq \phi$, go back to Step 3; otherwise stop.
```

5.1.2 Ancestor of a Split Activity

For a given start activity s, i.e. $s \in$ 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 *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 PreSplits($i \mid s$).

<u>Example 5-4. Ancestor of Split Activity</u>

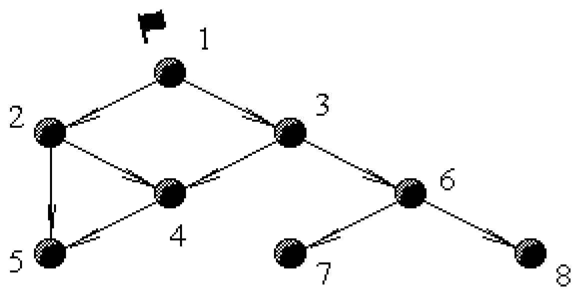


Figure 5-4. Ancestor of Split Activity

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

Sources(T) = {1}
Splits(T) = {1, 2, 3, 6}
Criticals(1→2) = ϕ, Criticals(1→3) = ϕ, Criticals(1→6) = {3}
PreSplits(1 | 1) = ϕ
PreSplits(2 | 1) = ({1} $\cup$ Criticals(1→2)) $\cap$ Splits(T) = {1}
PreSplits(3 | 1) = ({1} $\cup$ Criticals(1→3)) $\cap$ Splits(T) = {1}
PreSplits(6 | 1) = ({1} $\cup$ Criticals(1→6)) $\cap$ Splits(T) = {1, 3}

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; and both split activity 1 and activity 3 are ancestors of split activity 6.

5.1.3 Algorithm for <u>Getting Ancestors of a Split Activity</u>

This procedure is called by the algorithm for <u>Getting Ancestor Joins</u> (see Section 5.2.4).

<u>Hypothesis</u>
Updated Splits(T) keeps the set of split activities of a process definition (see Section 5.1.1). Updated Criticals(i→k), $\forall i$, $k \in T$, denotes the set of critical activities of paths i→k (see Section 4.2.3).

<u>Principle</u>
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. returns PreSplits($i \mid s$).
Temporary sets RestActivities, V are used here.

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

5.2 Join Activity

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

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

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

Assumption 5-1. Synchronization at a Join Activity
When a work is routed to a join activity, the join activity is not allowed to execute, if there is another work associated with the same process instance having the potential to flow to the join activity. So the work must wait at a join activity with the "*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 by the Espresso workflow engine.

From the assumption, at a join activity, several execution threads of a process instance will be joined into one execution thread. The assumption avoids workflow participants of the join activity to receive or execute a work associated with the same process instance several times from different predecessor activities. At run-time, the Espresso workflow engine checks regularly whether parallel work items waiting at join activities can be joined or further routed.

PM determines the join activities of a process definition according to the definition of a join activity. For the real world business requirements and circumstances, an activity can be specified as a never-join activity to avoid being determined as a join activity (see Example 5-14 in Section 5.2.3).

In order to comprehend the definition of a join activity, the definitions of a potential join activity and an ancestor join activity are further introduced.

5.2.1 Potential Join Activity

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

1. Paths($s \rightarrow i$) $\neq \phi$, or $i = s$;
2. Criticals($i \rightarrow j$) $= \phi$; and
3. if $(i, j) \in {}^{P}T(j)$, exists at least one q, otherwise exist at least two different q, $(q, j) \in {}^{P}T(j)$ and $q \neq i$, with
   Paths($i \rightarrow q$) $\neq \phi$ and $j \notin$ 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 without going 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 cannot be a join activity.

If an activity is critical on paths of $i \rightarrow j$, the activity can instead of activity j be a join activity of activity i (see Example 5-8 in Section 5.2.3).

5.2.2 Ancestor Join Activity

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 $\exists p$, $p \in$ PreSplits($i \mid s$) and $p \in$ Criticals($i \rightarrow q$) $\cup$ $\{q\}$ $\cup$ Criticals($q \rightarrow j$) with

$m \in$ Joins($p \mid s$) $\cap$ Reaches(j)

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

According to the definition of a join activity, 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 to activity j over the ancestor of split activity i (see Example 5-13 in Section 5.2.3).

5.2.3 Examples of Join Activities

The definition of a 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. Join Activity

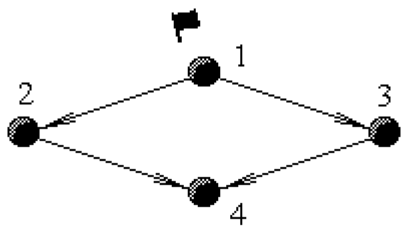


Figure 5-5. Split and Join

For the process definition in Figure 5-5,

Sources(T) = {1}
Splits(T) = {1}
PreSplits(1 | 1) = ϕ
Paths(1→2) = {(1, 2)}, Paths(1→3) = {(1, 3)}
Criticals(1→4) = ϕ, Criticals(1→2) = ϕ, Criticals(1→3) = ϕ
Joins(1 | 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→4) = ϕ; and
3. ${}^{P}T(4) = \{(2, 4), (3, 4)\}$, with
   Paths(1→2) = {(1, 2)} ≠ ϕ,
   Paths(1→3) = {(1, 3)} ≠ ϕ, and
   activity 4 is not critical on the paths 1→2 and 1→3; furthermore, because PreSplits(1 | 1) = ϕ, there is no ancestor join activity being critical on the paths.

At run-time, a process instance as well as an activity instance is created at activity 1. 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.

<u>Example 5-6. Multiple Join Activities</u>

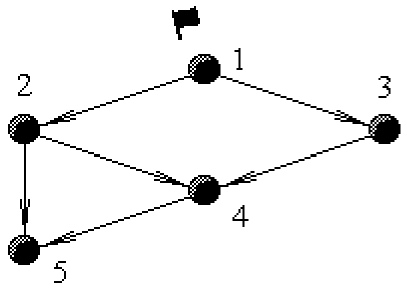


Figure 5-6. Multi-Split and Join

For the process definition in Figure 5-6,

Sources(*T*) = {1}
Splits(*T*) = {1, 2}
Joins(1 | 1) = {4, 5}, Joins(2 | 1) = {5}

That is, activity 1 is the start activity of the process definition, 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→4.

A process instance as well as an activity instance can be created at activity 1. At split activity 2, the activity instance, which becomes parallel after split activity 1 is completed, will result in two parallel activity instances of activity 4 and activity 5, after it is completed. Associated with a process instance of 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 instances of activity 2 and activity 3 associated with the same process instance are completed and joined together; and activity 5 can be executed only after parallel instances of activity 2 and activity 4 are completed and joined into one. When activity 5 is executed, there is only one activity instance within the process instance.

Example 5-7. Not a Join Activity

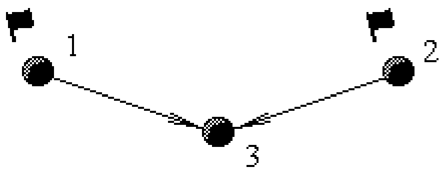


Figure 5-7. Not a Join—No Split

For the process definition in Figure 5-7,

Sources(T) = {1, 2}
Splits(T) = ϕ

Although activity 3 has two predecessors of activity 1 and activity 2, it is not a join activity. The work items coming 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. Not a Join Activity—Critical on Path

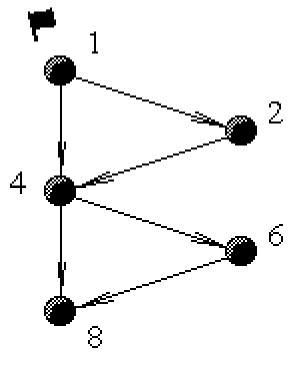


Figure 5-8. Not a Join—Critical on Path

For the process definition in Figure 5-8,

Sources(T) = {1}
Splits(T) = {1, 4}
Criticals(1→8) = {4}

Joins(1 | 1) = {4}, Joins(4 | 1) = {8}

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

Since Criticals(1→8) = {4} ≠ ϕ, activity 4, which is a join activity of split activity 1, is critical on paths of 1→8. Therefore, activity 8 is not a join activity of split activity 1.

Example 5-9. Not a Join Activity—Itself on Path

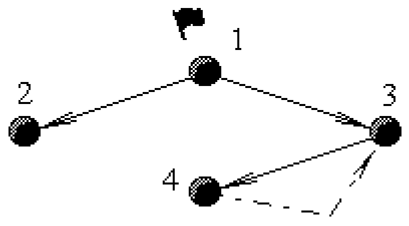


Figure 5-9. Not a Join—Itself on Path

For the process definition in Figure 5-9,

Sources(T) = {1}
Splits(T) = {1}
Criticals(1→4) = {3}
Joins(1 | 1) = ϕ

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∈Criticals(1→4) (activity 3 is on one of the distinct paths (1, 3, 4, 3)).

Example 5-10. Join Activity of Itself

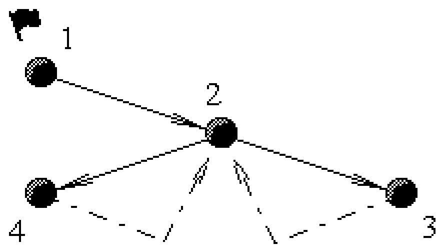


Figure 5-10. Join at Split

For the process definition in Figure 5-10,

Sources(T) = {1}
Splits(T) = {2}

Joins(2 | 1) = {2}

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 may flow back to activity 2 and be joined together there.

Example 5-11. Join Activity—Start Activity on Path

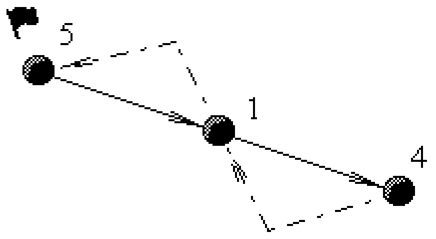


Figure 5-11. Join—Start on Path

For the process definition in Figure 5-11,

Sources(*T*) = {5}
Split(*T*) = {1}
Joins(1 | 5) = {1}

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 of 1→1.

A process instance can be 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 may exist. Thus, 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. Join Activity—Ancestor on Path
For the process definition in Figure 5-12,

Sources(*T*) = {1}
Splits(*T*) = {1, 3}
PreSplits(3 | 1) = {1}
Joins(1 | 1) = {4}, Joins(3 | 1) = {4}

Activity 4 is also 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}.

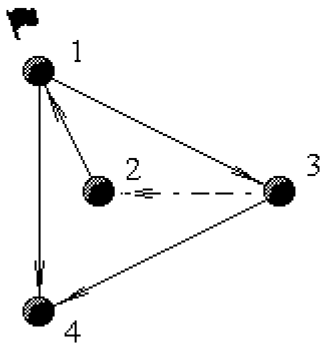


Figure 5-12. Join—Ancestor on Path

A work going 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. Not a Join Activity—Ancestor Join Activity

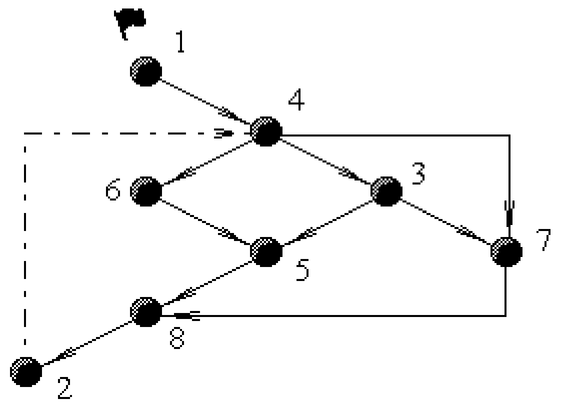


Figure 5-13. Not a Join—Ancestor Join

For the process definition in Figure 5-13,

Sources(*T*) = {1}
Splits(*T*) = {3, 4}
Criticals(3→6) = {2, 4, 8}, Criticals(3→4) = {2, 8}
PreSplits(3 | 1) = {4}
Joins(4 | 1) = {5, 7, 8}
PreJoins(3→6→5 | 1) = {5, 7, 8}, PreJoins(3→4→7 | 1) = {5, 7, 8}
Joins(3 | 1) = {8}

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 Criticals(3→4) = {2, 8} ≠ ϕ.

Activity 5 is not a join activity of split activity 3, because on the path of 3→5 over activity 6 (i.e. path (3, 7, 8, 2, 4, 6, 5)), one of the two distinct paths of 3→5, ancestor join activity 8 belongs to Criticals(3→6). Similarly, activity 7 is not a join activity of split activity 3, because on the path 3→7 over activity 4 (i.e. path (3, 5, 8, 2, 4, 7)), one of the two distinct paths of 3→7, ancestor join activity 8 belongs to Criticals(3→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 neither joined at activity 5 nor activity 7, but at activity 8.

Example 5-14. Never-Join Activity

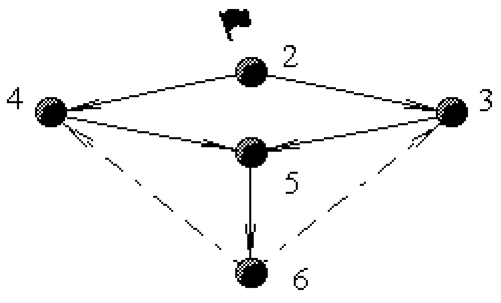


Figure 5-14. Never-Join Activity

For the process definition in Figure 5-14, activities 3, 4 and 5 can be determined as join activities by PM according to the definition of a join activity. At start activity 2, a process instance as well as an activity instance is created. After activity 2 is completed, the activity instance is split into two: one at activity 3 and another at activity 4. The work at activity 3 cannot be executed at once, because it waits for joining with the work that may come from activity 6 via path (2, 4, 5, 6, 3). Meanwhile, the work at activity 4 waits also for joining.

If the process designer wants that the parallel activity instances split at activity 2 be only joined at activity 5, he should specify that activities 3 and 4 are never-join activities.

5.2.4 Algorithm for Getting Ancestor Joins

This procedure is called by the algorithm for Determining Joins (see Section 5.2.5).

Hypothesis

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

PreSplits($k \mid s$) denotes a set of ancestors of split activity k from start activity s. Joins($p \mid s$), $p \in$ Splits(T), represents a set of join activities of split activity p relevant to start activity s.

Principle

Activities i, q, j and s are parameters of the procedure. Here $s \in$ Sources(T) and $i \in$ Splits(T); activity j is dealt with to determine whether it is 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. It is known that Joins($p \mid s$), $\forall p \in$ PreSplits($i \mid s$), has been determined.

Returned set 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 RestSplits, V are used.

Procedure (i, q, j, s)

```
Step 1: PreJoins ← $\phi$;
Step 2: $V$ ← Criticals($i \rightarrow q$) $\cup$ $\{q\}$ $\cup$ Criticals($q \rightarrow j$);
Step 3: RestSplits ← Getting Ancestors of a Split Activity ($i$, $s$) (i.e.
        PreSplits($i \mid s$));
Step 4: if RestSplits = $\phi$, stop (return PreJoins);
Step 5: remove an element, say split activity $p$, from set 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: PreJoins ← PreJoins $\cup$ (Joins($p \mid s$) $\cap$ Reaches($j$));
Step 8: go to Step 4.
```

5.2.5 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. Sources(T) is the set of start activities. ${}^{P}T(j)$, $j \in T$, represents the set of predecessors of activity j. Splits(T) stands for the set of split activities of the process definition. Updated

Reaches(k), $\forall k \in T$, is a set of activities to which activity k is reachable. Criticals($i \rightarrow k$), $i, k \in T$, is the set of critical activities of paths $i \rightarrow k$.

Principle

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

Temporary set RestActivities keeps the activities that have not been determined whether to be a join activity of activity i. Set RestLinks keeps not treated predecessors of current considered activity. Temporary variable CountPaths keeps current number of paths meeting the definition for determining a join activity of split activity i.

Set U is used Temporarily.

Procedure (i, s)

```
Step 1:  Joins($i \mid s$) $\leftarrow \phi$;
Step 2:  if $i = s$, go to Step 4;
Step 3:  if $i \notin$ Reaches($s$) (paths of $s \rightarrow i$ do not exist), stop;
Step 4:  RestActivities $\leftarrow$ Reaches($i$);
Step 5:  remove an element, say activity $j$, from RestActivities;
Step 6:  if $|^{P}T(j)| < 2$ (activity $j$ has no or only one predecessor) or activity $j$ is a
         never-join activity, go to Step 19;
Step 7:  CountPaths $\leftarrow 0$;
Step 8:  RestLinks $\leftarrow {}^{P}T(j)$;
Step 9:  remove an element, say link $(q, j)$, from set RestLinks;
Step 10: if $q = i$ (activity $i$ is connected to activity $j$ by a link), go to Step 15;
Step 11: if $q \notin$ Reaches($i$) (paths $i \rightarrow q$ do not exist), go to Step 16;
Step 12: $U \leftarrow$ Getting Ancestor Joins with parameters $i$, $q$, $j$ and $s$; $U \leftarrow U \cup \{j\}$;
Step 13: if $q \in U$ (activity $q$ is an ancestor join activity), go to Step 16;
Step 14: if an activity in set $U$ belongs to Criticals($i \rightarrow q$), go to Step 16;
Step 15: CountPaths $\leftarrow$ CountPaths + 1; if CountPaths > 1, go to Step 18;
Step 16: if RestLinks $\neq \phi$, go back to Step 9;
Step 17: if CountPaths < 2, go to Step 19;
Step 18: Joins($i \mid s$) $\leftarrow$ Joins($i \mid s$) $\cup \{j\}$;
Step 19: if RestActivities $\neq \phi$, go back to Step 5; otherwise stop.
```

5.3 Parallel Activity

Activity k is a *parallel activity*, if $\exists i, j \in T - \{k\}$ and $s \in$ Sources(T), with

1. $j \in$ Joins($i \mid s$);
2. Paths($i \rightarrow k$) $\neq \phi$, Paths($k \rightarrow j$) $\neq \phi$; and

3. $j \notin$ Criticals($i \to k$) and $i \notin$ Criticals($k \to j$).

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 *Parallels*($i \to j \mid s$).

Example 5-15. Parallel Activity
For the process definition in Figure 5-15,

Sources(T) = {1}	
Splits(T) = {1, 2}	
Joins(1 | 1) = {3},	Joins(2 | 1) = ϕ
Criticals(1→5) = {3},	Criticals(4→3) = {1}
Parallels(1→3 | 1) = {2, 6}	

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

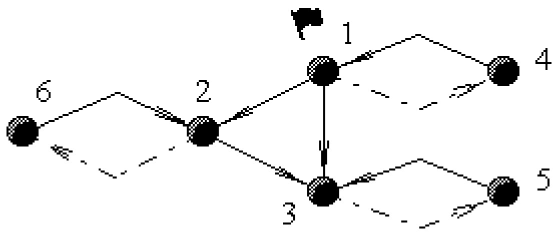


Figure 5-15. Parallel Activity

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 either, because join activity 3 belongs to Criticals(1→5).

It is logical that 5$\notin$Parallels(1→3 | 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$\notin$Parallels(1→3 | 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. In the Espresso WfMS, 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 *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, 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. It is called to-join list of activity k.

If activity k is a parallel activity, ToJoins(k) has at least one element, i.e. ToJoins(k) $\neq \phi$. If ToJoins(k) is not empty, a work at activity k has at run-time the potential to be joined at the join activities belonging to the set. ToJoins(k), $\forall k \in T$, is used by the Espresso workflow engine to decide joining of work items associated with the same process instance.

Assumption 5-2. To-Join List (Run-time)
If $j \in$ ToJoins(i), $i \in T$, work at activity i has the potential to be joined at join activity j.

Example 5-16. To-Join List
For the process definition in Figure 5-5 of Section 5.2.3, it is known that

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

From the definitions of a parallel activity and the to-join list, we have

$$\text{Parallels}(1 \rightarrow 4 \mid 1) = \{2, 3\}$$

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 determined workflow control data as shown in Figure 5-16. "IsJoin" beside each activity icon indicates whether the

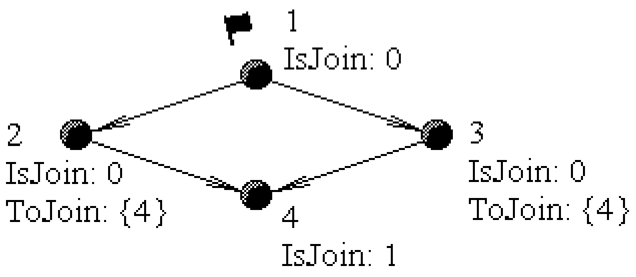


Figure 5-16. To-Join List

activity is a join activity or not, and “ToJoin” presents the set of join activities where the work at the activity has the potential to join.

In the Espresso WfMS, when a work associated with a process instance comes to activity 4, it will be examined whether the work must 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. Therefore, 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 deadlock situations (see Chapter 6). They are calculated by PM and saved with activity definitions in the Espresso 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 in Section 5.3, 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. Therefore, 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 the definition of a join activity 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. Sources(T) denotes the set of start activities. Splits(T) stands for the set of split activities. Updated Reaches(i), $\forall i \in T$, is the set of activities which are reachable from activity i. 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 $U \subseteq$ Splits(T) and $s \in$ Sources(T). This procedure returns a split activity i, $i \in U$, if

$i = s$ or
Paths($s \rightarrow i$) $\neq \phi$ and $\forall p \in$ PreSplits($i \mid s$), with $p \notin U$

otherwise returns NULL.
Temporary set RestSplits is used.

Procedure (U, s)
```
Step 1:  RestSplits ← $U$;
Step 2:  if RestSplits = $\phi$ (required split activity cannot be found), stop (return
         NULL);
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 $i \notin$ Reaches($s$) (paths $s \rightarrow i$ do not exist), go to Step 2;
Step 6:  if RestSplits = $\phi$, 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 $p$);
Step 9:  if $p \notin$ Reaches($s$) (paths of $s \rightarrow p$ do not exist), go to Step 6;
Step 10: if $p \notin$ 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 Section 6.2.1.

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 on, will be individually dealt with for getting all the 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*), $\forall(k, n)\in T^S(k)$ with $n\in$ Parallels($i\rightarrow j \mid s$).

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, ..., 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.

<u>Main Procedure</u>

```
Step 1:  call Determining Reachability and Criticality (Reaches($i$), $i\in T$, as well
         as Criticals($i\rightarrow k$), $\forall k\in$ 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 $i$ = NULL (no split activity reachable from activity $s$ can be found in
         set $V$), 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 $\leftarrow$ 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 $j$ (determine parallel activities between split
         activity $i$ and the join activity $j$ relevant to start activity $s$);
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.
```

<u>Sub Procedure (*s*, *i*, *k*, *j*)</u>

```
Step 1:  RestLinks $\leftarrow T^S(k)$;
Step 2:  if RestLinks = ϕ, stop;
Step 3:  remove an element, say link (k, n), from set RestLinks;
Step 4:  if n = j (n is the join activity), go to Step 2;
Step 5:  if n is in PathStack() or n∈ParallelActivities (n has been dealt with),
         go to Step 2;
Step 6:  if j∉Reaches(n) (paths of n→j do not exist), go to Step 2;
Step 7:  U ← {i, j};
Step 8:  if n∈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 Criticals(n→j) (split activity i or join
         activity j is critical on paths n→j), go to Step 2;
Step 10: (activity n is a parallel activity between split activity i and the join
         activity j relevant to start activity s) let
             CopyFlag(n) ← TRUE
             ToJoins(n) ← ToJoins(n) ∪ {j}
             ParallelActivities ← ParallelActivities ∪ { n }
Step 11: StackPointer ← StackPointer + 1; PathStack(StackPointer) ← n;
Step 12: call the sub procedure self with parameters s, i, n and j;
Step 13: StackPointer ← StackPointer – 1; go to Step 2.
```

Example 5-17. Workflow Control Data

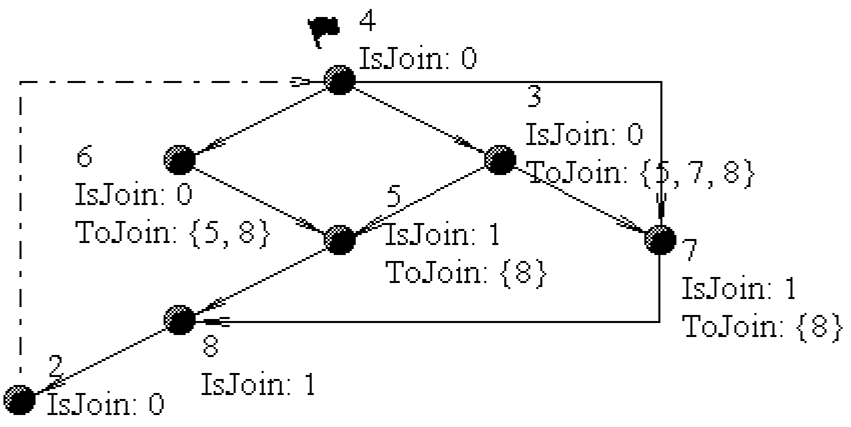


Figure 5-17. Workflow Control Data

For the process definition in Figure 5-17, activity 4 is specified as a start activity. CopyFlag(i), JoinFlag(i) and ToJoins(i), $i \in T$, are determined by the algorithm and the results are displayed beside each activity icon. That is,

ToJoins(2) = ToJoins(4) = ToJoins(8) = ϕ
ToJoins(3) = {5, 7, 8}
ToJoins(5) = ToJoins(7) = {8} and
ToJoins(6) = {5, 8}

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 5-17 by specifying merely activity 3 a start activity as shown in Figure 5-18 and get the

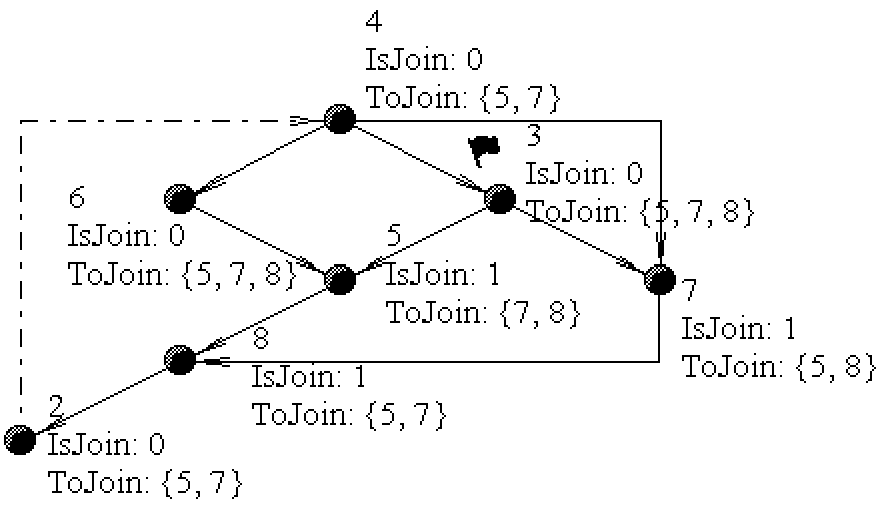


Figure 5-18. Workflow Control Data—Change Start

following results

ToJoins(2) = ToJoins(4) = ToJoins(8) = {5, 7}
ToJoins(3) = ToJoins(6) = {5, 7, 8}
ToJoins(5) = {7, 8} and
ToJoins(7) = {5, 8}

As a consequence, 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 yield quite different effects of joining circumstances.

5.5 Conclusion

If one execution thread of a process instance is possible to split into multiple concurrent execution threads after execution of activity *i*, activity *i* is a split activity, denoted by $i \in \text{Splits}(T)$. If it can happen at activity *j* that multiple execution threads split at 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 going over split activity *p*, activity *p* is called an ancestor of split activity *i*, denoted by *p*∈PreSplits(*i*|*s*). Join activity *m* of split activity *p* relevant to start activity *s* is an ancestor join activity on a path of *i*→*q*→*j*, denoted by *m*∈PreJoins(*i*→*q*→*j*|*s*), if join activity *m* is reachable from activity *j*, and split activity *p*, which is an ancestor of split activity *i* relevant to start activity *s*, is identical with activity *q* or is critical on paths *i*→*q* or *q*→*j*. Ancestor join activities are used for determining whether potential join activities are join activities of a process definition. If an activity is specified as a never-join activity, it will not be determined as a join activity by PM.

Activity *k* is a parallel activity between split activity *i* and join activity *j* relevant to start activity *s*, 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 the parallel activities is denoted by *Parallels*(*i*→*j*|*s*). The set of join activities to which parallel activity *k* has a path and is reachable from the split activity of a related join activity is denoted by ToJoins(*k*), and called to-join list.

Workflow control data are determined by PM. They indicate whether an activity is a parallel activity and/or a join activity, and what is the to-join list of the activity. The Espresso workflow engine uses workflow control data to control the flowing of work items in accordance with the process definition. Figure 5-19 illustrates how the workflow control data are determined.

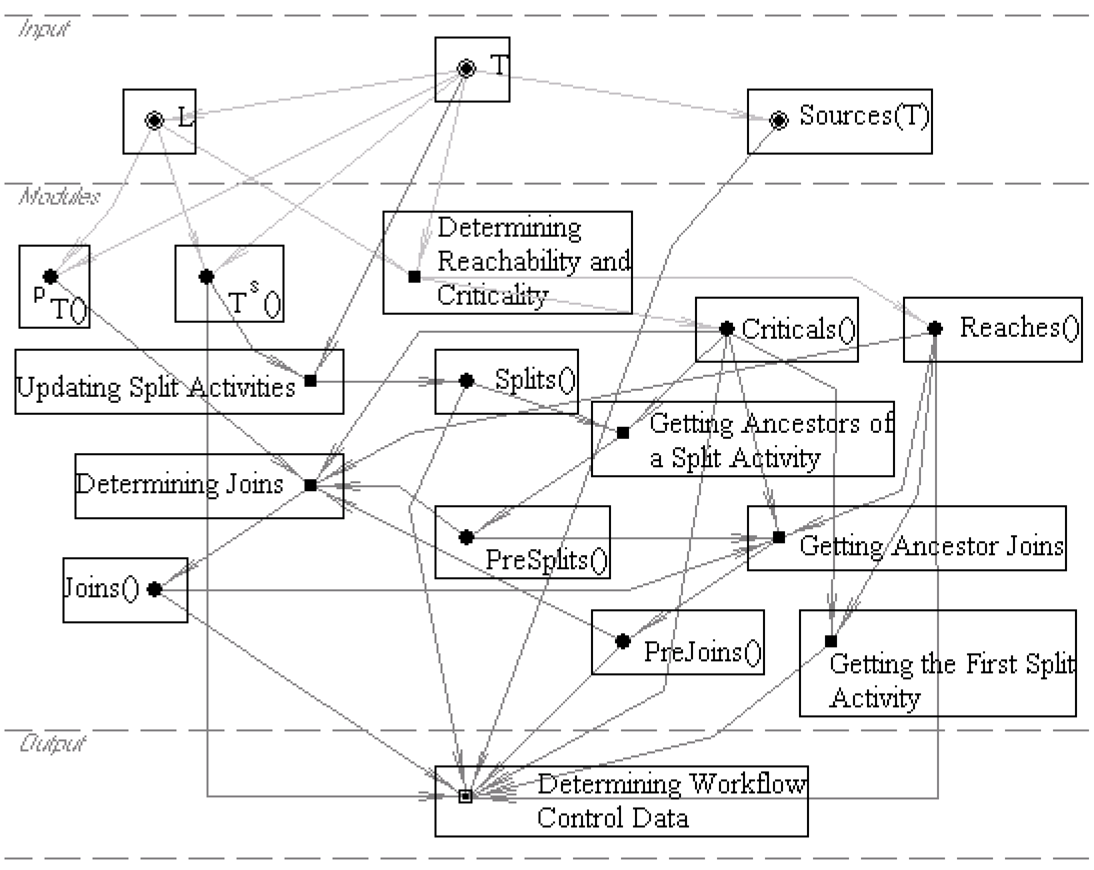

Input
T
L
Sources(T)
Modules
Determining Reachability and Criticality
$^pT()$
$T^s()$
Criticals()
Reaches()
Updating Split Activities
Splits()
Getting Ancestors of a Split Activity
Determining Joins
PreSplits()
Getting Ancestor Joins
Joins()
PreJoins()
Getting the First Split Activity
Output
Determining Workflow Control Data

Figure 5-19. Determining Workflow Control Data

6 DEADLOCK

A deadlock cycle modeled in a process definition can yield a deadlock situation of a process instance. Join priorities specified to the join activities involved in a deadlock cycle are used by the Espresso workflow engine to release the deadlock situation.

6.1 Deadlock Situation and Deadlock Cycle

According to Assumption 5-1 in Section 5.2 and Assumption 5-2 in Section 5.3, if a process definition has more than one join activity, a deadlock situation of a process instance may arise at run-time among the join activities. The work at a join activity cannot be executed if at other activities there are work items associated with the same process instance and having the potential to flow to the join activity. Parallel work items at different join activities may wait for one another. Thus, the process instance cannot run further.

Example 6-1. Deadlock Situation

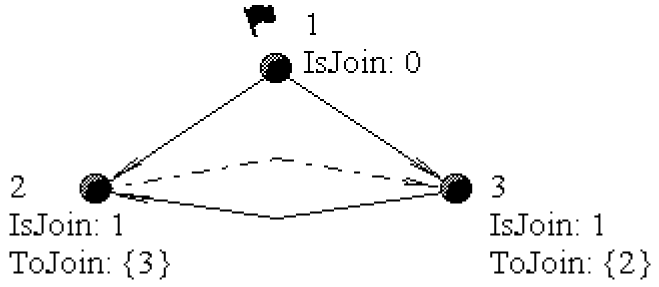


Figure 6-1. Deadlock

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 cannot be executed while another work associated with the same process instance is at activity 3 and has the potential to come to activity 2. And meanwhile, the work arriving at activity 3 is waiting also for the work at activity 2 flowing to activity 3 for joining. Therefore, the parallel work items at activity 2 and activity 3 are waiting for each other and cannot be executed. Thus, the process instance cannot run further and a deadlock situation arises.

Two or more join activities within a process definition can yield at run-time a *deadlock situation* of a process instance in accordance with the process definition. There are m ($m > 1$) different join activities j_k, $j_k \in T$, $k = 1, 2, \ldots, m$, involved in a deadlock, if

$$j_m \in \text{ToJoins}(j_1) \text{ and}$$
$$j_k \in \text{ToJoins}(j_{k+1}),\ k = 1, 2, \ldots, m - 1$$

That is, there is a *deadlock cycle* in 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.

According to the definition, a deadlock cycle is a set of the join activities that combine together a reachable cycle through the to-join lists.

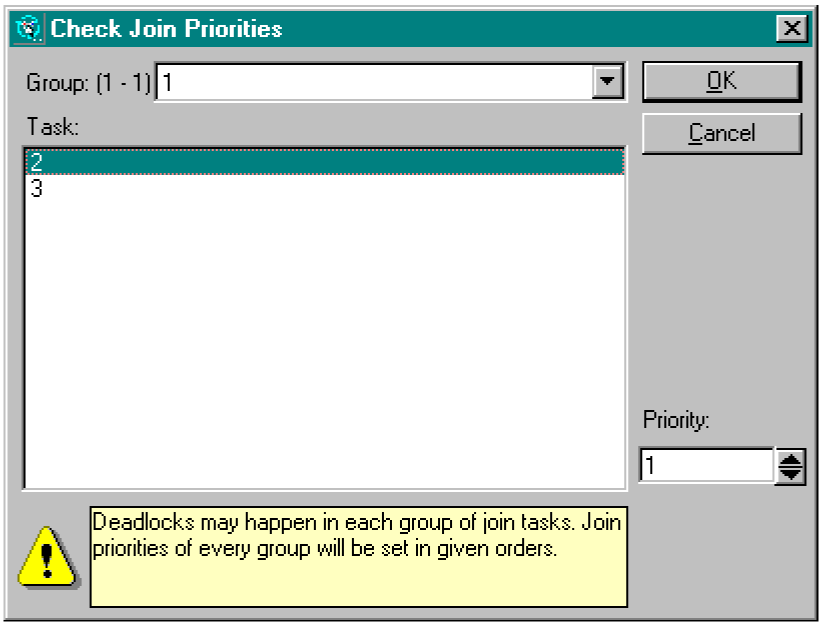
6.2 Join Priority and Deadlock Release

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 deadlock situations.

Assumption 6-1. Deadlock Release
To release a deadlock situation in the Espresso WfMS, 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 of activity i, denoted by *Priority*(i), $i \in T$, orders joining preference of the activity. The default value 0 stands for the lowest joining chance. When a deadlock situation among join activities is detected, the join activity with the lowest value of join priority will be executed at once without joining. Join priorities of all the join activities involved in a deadlock must be different from one another. Thus, if an activity is involved in several deadlock cycles, these deadlock cycles will be grouped together for the assignment of different join priorities to the activities in the group.

Example 6-2. Join Priority
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 dialogue box "Check Joining Priorities" as shown in Figure 6-2 and can be modified by the process designer. In the activity (task) list, the first activity has priority 1, the second 2, and so on. Join priority specification of join activities in a group is independent on that in another group.



Group: (1 - 1) 1

Task:

2
3

Priority:

1

Deadlocks may happen in each group of join tasks. Join priorities of every group will be set in given orders.

Figure 6-2. Join Priority Specification

Suppose that Priority(2) = 1 and Priority(3) = 2 for the process definition. In the Espresso WfMS, when a work waits at activity 2 for joining with one coming 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 coming from activity 2 (there is a deadlock situation between join activities 2 and 3), the join priorities are used now. Because activity 2 has a lower value of join priority than activity 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 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 the deadlock cycles of a process definition.

Hypothesis

T denotes the activity set of a process definition. JoinFlag(*i*) stands for whether activity *i* is a join activity or not, and ToJoins(*i*) keeps the set of join activities where a work at activity *i* has the potential to join, $\forall i \in T$.

Principle

A set of elementary deadlock cycles 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, ..., StackPointer. StackPointer is assigned with 0, before the procedure is called by the main procedure. Join activity j, $j \in$ ToJoins(i), must not be included in the searching path, if $i\#j \notin U$. It is already known that PathStack(1)$\in$ ToJoins(k). Therefore, as a result, the found path of join activities yields a deadlock cycle and will be added to set Deadlocks, which keeps the different deadlock cycles in a process definition.

Variable MaxDeadlocks keeps the maximum number of join activities able to be involved in a deadlock cycle. Since the deadlock cycles will be grouped for the specification of join priorities afterwards, the procedures will be stopped when the number of join activities involved in the newly found deadlock cycle is equal to MaxDeadlocks.

Temporary sets U, RestActivities are used here.

```
Main Procedure
Step 1:  call Determining Workflow Control Data (determine JoinFlag($i$) and
         ToJoins($i$), $\forall i \in T$);
Step 2:  Deadlocks $\leftarrow \phi$;
Step 3:  $U \leftarrow \phi$; MaxDeadlocks $\leftarrow$ 0;
Step 4:  RestActivities $\leftarrow T$;
Step 5:  if RestActivities = $\phi$, go to Step 9;
Step 6:  remove an element, say activity $i$, from set RestActivities;
Step 7:  if JoinFlag($i$) = TRUE and ToJoins($i$) $\neq \phi$ (activity $i$ is a join activity
         and may be involved in deadlock cycles), let
             $U \leftarrow U \cup \{i\#j\}$, $\forall j \in$ ToJoins($i$)
             MaxDeadlocks $\leftarrow$ MaxDeadlocks + 1
Step 8:  go to Step 5;
Step 9:  if $|U| \leq 1$ (none or one pair of join activities cannot yield a deadlock
         cycle), stop (return set Deadlocks);
Step 10: remove an element, say join activity pair $i\#j$, from set $U$;
Step 11: StackPointer $\leftarrow$ 0; IsContinue $\leftarrow$ call the sub procedure with parameters
         $j$, $i$ and $U$;
Step 12: if IsContinue = TRUE, go to Step 9; otherwise stop (return set
         Deadlocks).

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),
            1° add the sequence of join activities on the searching path (i.e.
               PathStack(1), PathStack(2), ..., PathStack(StackPointer)) to set
               Deadlocks;
```

```
          2° if StackPointer < MaxDeadlocks, go to Step 9; otherwise stop
             (return FALSE);
Step 3: RestActivities ← ToJoins(i);
Step 4: if RestActivities = ϕ, go to Step 9;
Step 5: remove an element, say join activity j, from set RestActivities;
Step 6: if $i\#j\notin U$, go to Step 4;
Step 7: if j = PathStack(n) with $n\in$ [1, StackPointer] (join activity j is on current
        searching path), go to Step 4;
Step 8: IsContinue ← call the sub procedure self with parameters j, k and U; if
        IsContinue = TRUE, go to Step 4, otherwise stop (return FALSE);
Step 9: StackPointer ← StackPointer – 1; stop (return TRUE).
```

6.2.2 Algorithm for Grouping Deadlocks

Deadlock cycles of a process definition will be grouped for specifying join priorities and for reporting verification results.

Hypothesis
A deadlock cycle is a set of the join activities that may yield at run-time a deadlock situation.

Principle
This procedure returns a set of join activity groups generated via union of deadlock cycles having a common join activity. The returned value is kept in set GroupDeadlocks.

Temporary sets RestDeadlocks, CurrentGroup, and NoRelations are used in the procedure. Set RestDeadlocks keeps the deadlock cycles that have not been grouped. Set NoRelations contains deadlock cycles that have been removed from RestDeadlocks but have not been grouped in set CurrentGroup, the just treated deadlock group. Variable ToAddGroup means whether to initialize set CurrentGroup with an element in set RestDeadlocks. If not, CurrentGroup may have been grouped with some new join activities and will be dealt with further.

Procedure

```
Step 1:  GroupDeadlocks ← ϕ;
Step 2:  RestDeadlocks ← Getting Elementary Deadlocks;
Step 3:  ToAddGroup ← TRUE;
Step 4:  if RestDeadlocks = ϕ, 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 cycle CurDeadlock, from
         RestDeadlocks; CurrentGroup ← CurDeadlock;
Step 7:  add CurrentGroup in set GroupDeadlocks (CurrentGroup in set
         GroupDeadlocks can be changed);
```

```
Step 8:  (to try to group deadlock cycles in set RestDeadlocks with
         CurrentGroup) ToAddGroup $\leftarrow$ TRUE;
Step 9:  NoRelations $\leftarrow \phi$;
Step 10: if RestDeadlocks = $\phi$, go to Step 15;
Step 11: remove an element, say deadlock cycle CurDeadlock, from set
         RestDeadlocks;
Step 12: if CurrentGroup $\cap$ CurDeadlock $\neq \phi$ (at least one join activity in
         deadlock cycle 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: RestDeadlocks $\leftarrow$ NoRelations; go to Step 4.
```

<u>Example 6-3. Group Deadlock Cycles</u>

For the process definition in Figure 6-3, all activities are join activities (see the workflow control data beside each activity icon). The algorithm

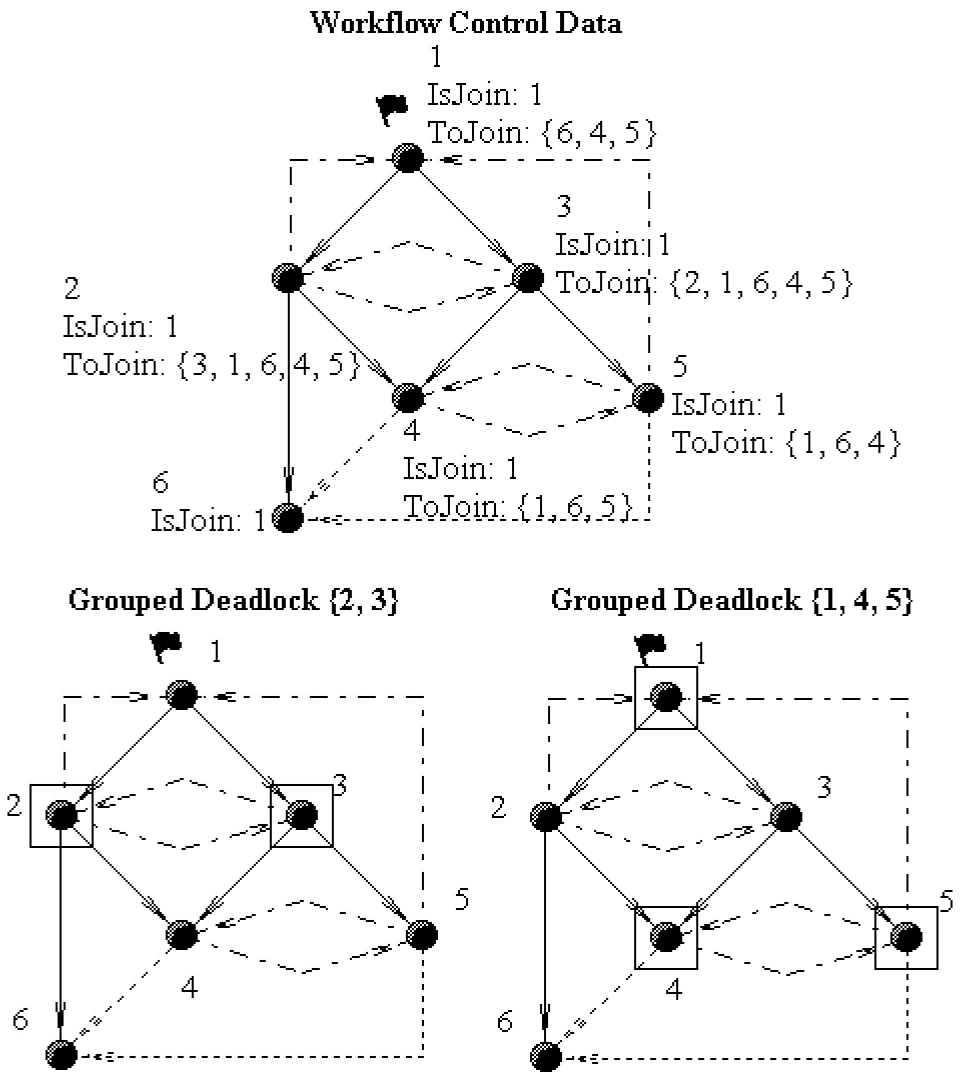


Figure 6-3. Group Deadlocks

for Getting Elementary Deadlocks finds the following deadlock cycles:

4→1→(4),	4→5→1→(4),	5→1→(5)
5→4→1→(5),	3→2→(3),	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 each of 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 Choice" links. Activities 3, 4, and 6 are join activities. The algorithm for

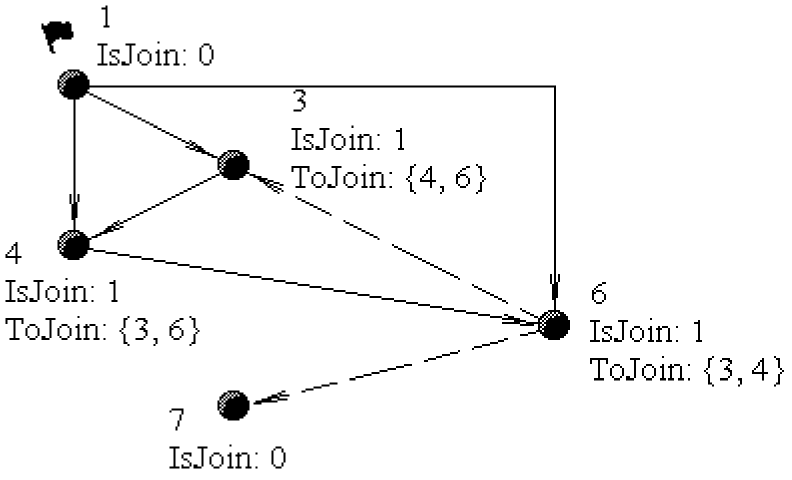


Figure 6-4. Deadlock Release

Getting Elementary Deadlocks will find the following deadlock cycles:

4→3→(4), 4→6→3→(4), 6→3→(6), 6→4→3→(6) and 6→4→(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.

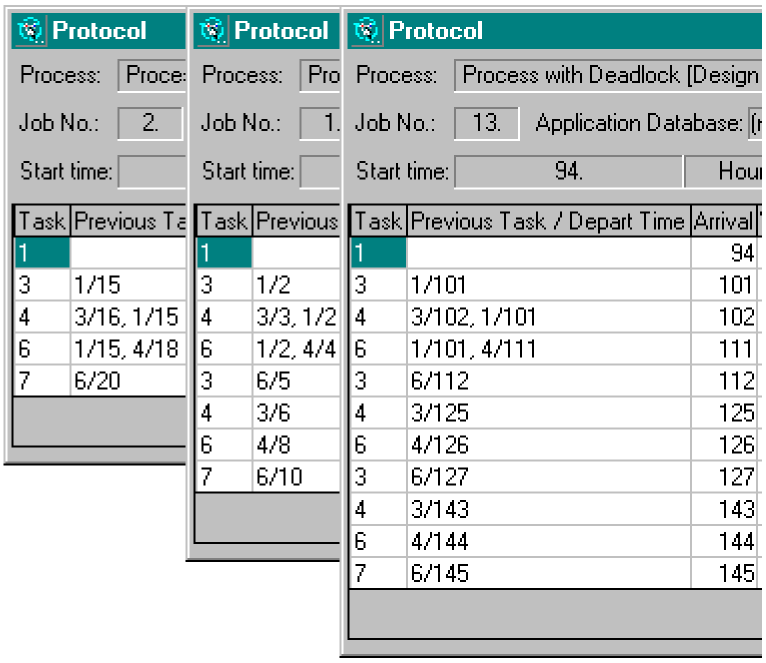
The following examples explain how different specifications of join priorities influence the activity (task) execution threads of the process instances (jobs) in accordance with the process definition. The process protocol in a table presents an execution thread of a process instance from the start activity to the last activity (see Section 13.3.1). 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

Priority(3) = 1, Priority(4) = 2 and Priority(6) = 3

Process instances generated in the Espresso WfMS can have the activity execution threads like those shown in Figure 6-5. After activity 1 is completed,



Process: Proce

Job No.: 2.

Start time:

Task	Previous Ta
1	
3	1/15
4	3/16, 1/15
6	1/15, 4/18
7	6/20

Process: Pro

Job No.: 1.

Start time:

Task	Previous
1	
3	1/2
4	3/3, 1/2
6	1/2, 4/4
3	6/5
4	3/6
6	4/8
7	6/10

Process: Process with Deadlock [Design

Job No.: 13. Application Database: [r

Start time: 94. Hou

Task	Previous Task / Depart Time	Arrival
1		94
3	1/101	101
4	3/102, 1/101	102
6	1/101, 4/111	111
3	6/112	112
4	3/125	125
6	4/126	126
3	6/127	127
4	3/143	143
6	4/144	144
7	6/145	145

Figure 6-5. Deadlock Release: 3→4→6

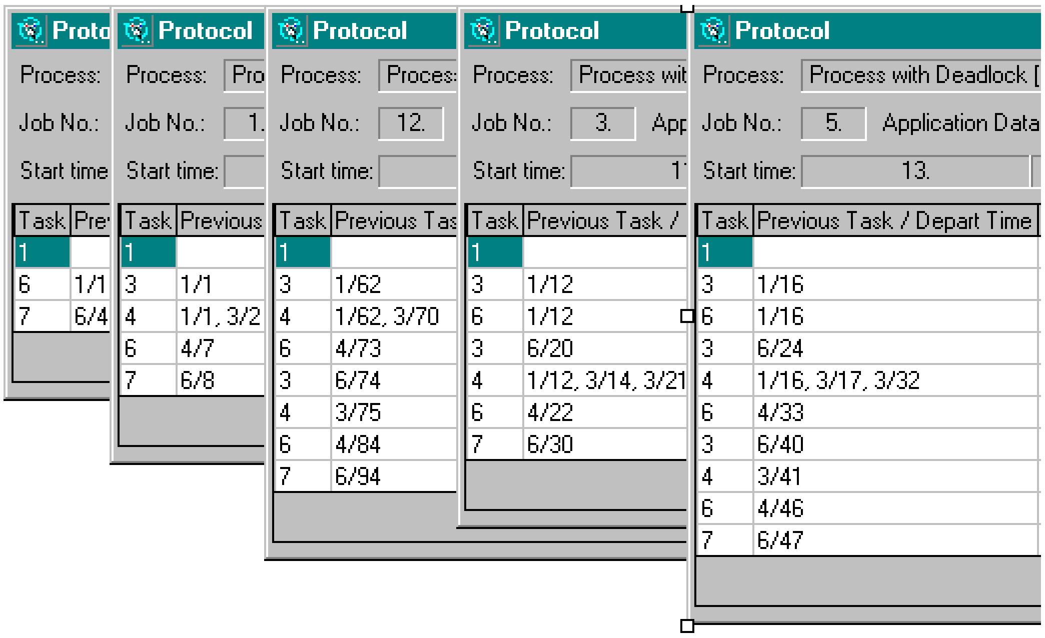
the work is split into three pieces that are sent to activities 3, 4 and 6 separately. Now a deadlock situation of the process instance arises. According to the join priority specification, activity 3 can be executed without joining. After activity 3 is completed, the work is sent to activity 4 and is joined to the work coming 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 coming 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

Priority(3) = 1, Priority(6) = 2 and Priority(4) = 3

The execution threads of a process instance can be one of those shown in Figure 6-6.



Proto

Process:

Job No.:

Start time

Task	Pre
1	
6	1/1
7	6/4

Protocol

Process: Pro

Job No.: 1.

Start time:

Task	Previous
1	
3	1/1
4	1/1, 3/2
6	4/7
7	6/8

Protocol

Process: Proces

Job No.: 12.

Start time:

Task	Previous Tas
1	
3	1/62
4	1/62, 3/70
6	4/73
3	6/74
4	3/75
6	4/84
7	6/94

Protocol

Process: Process wit

Job No.: 3. App

Start time: 1

Task	Previous Task /
1	
3	1/12
6	1/12
3	6/20
4	1/12, 3/14, 3/21
6	4/22
7	6/30

Protocol

Process: Process with Deadlock [

Job No.: 5. Application Data

Start time: 13.

Task	Previous Task / Depart Time
1	
3	1/16
6	1/16
3	6/24
4	1/16, 3/17, 3/32
6	4/33
3	6/40
4	3/41
6	4/46
7	6/47

Figure 6-6. Deadlock Release: 3→6→4

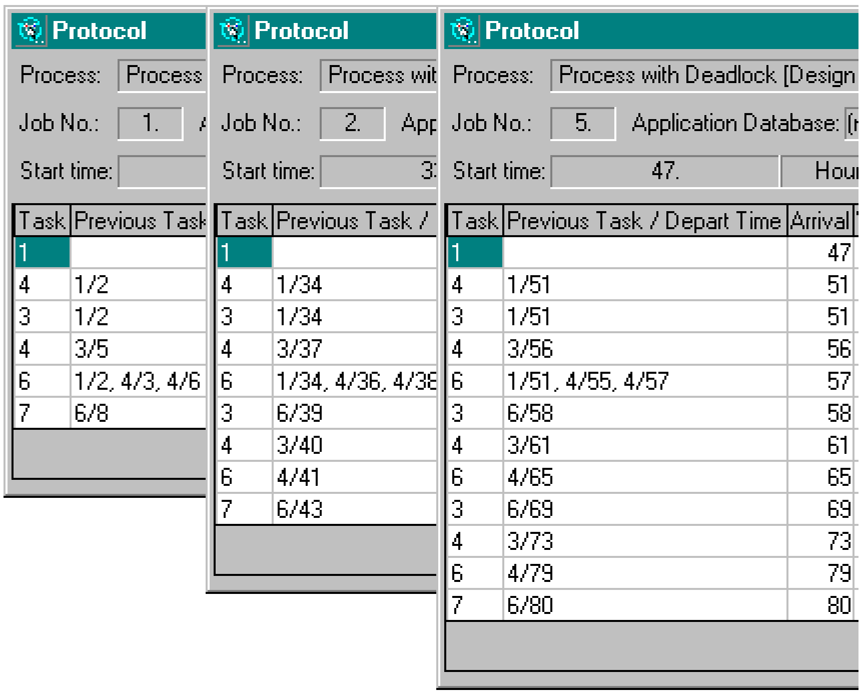
When three parallel work items are waiting at join activities 3, 4 and 6, activity 3 can be executed without joining and then the work is sent to activity 4 after it is completed at activity 3. As it is joined at activity 4 with the work from activity 1, activity 6 can be executed without joining. After 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 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 chance 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 activity execution threads of a process instance can be those as shown in Figure 6-7.



Protocol

Process: Process

Job No.: 1. A

Start time:

Task	Previous Task
1	
4	1/2
3	1/2
4	3/5
6	1/2, 4/3, 4/6
7	6/8

Protocol

Process: Process wit

Job No.: 2. App

Start time: 3

Task	Previous Task /
1	
4	1/34
3	1/34
4	3/37
6	1/34, 4/36, 4/3E
3	6/39
4	3/40
6	4/41
7	6/43

Protocol

Process: Process with Deadlock [Design

Job No.: 5. Application Database: [r

Start time: 47. Hou

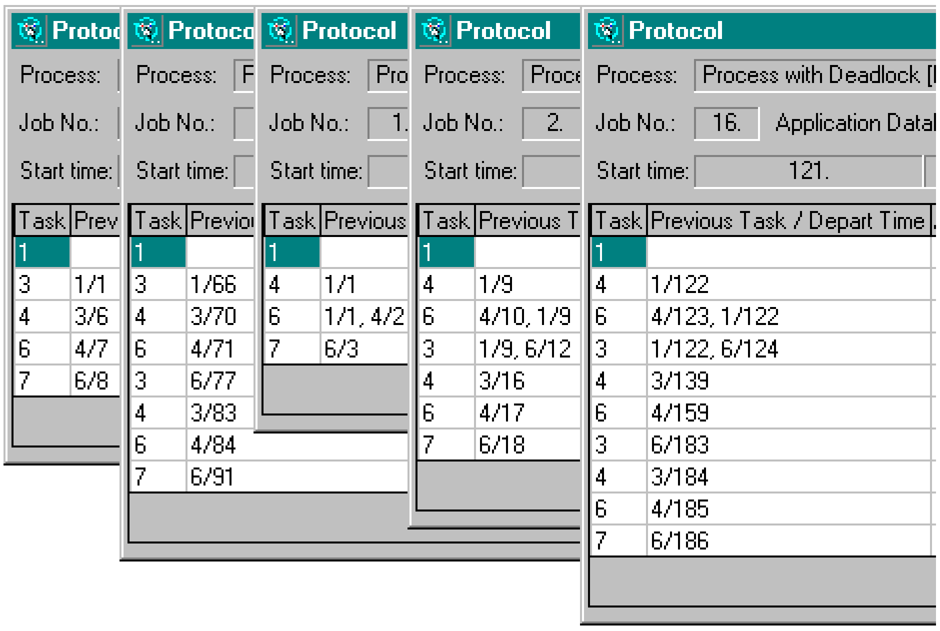
Task	Previous Task / Depart Time	Arrival
1		47
4	1/51	51
3	1/51	51
4	3/56	56
6	1/51, 4/55, 4/57	57
3	6/58	58
4	3/61	61
6	4/65	65
3	6/69	69
4	3/73	73
6	4/79	79
7	6/80	80

Figure 6-7. Deadlock Release: 4→3→6

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



Protoc

Process:

Job No.:

Start time:

Task	Prev
1	
3	1/1
4	3/6
6	4/7
7	6/8

Protoco

Process: F

Job No.:

Start time:

Task	Previo
1	
3	1/66
4	3/70
6	4/71
3	6/77
4	3/83
6	4/84
7	6/91

Protocol

Process: Pro

Job No.: 1.

Start time:

Task	Previous
1	
4	1/1
6	1/1, 4/2
7	6/3

Protocol

Process: Proce

Job No.: 2.

Start time:

Task	Previous T
1	
4	1/9
6	4/10, 1/9
3	1/9, 6/12
4	3/16
6	4/17
7	6/18

Protocol

Process: Process with Deadlock [I

Job No.: 16. Application Data

Start time: 121.

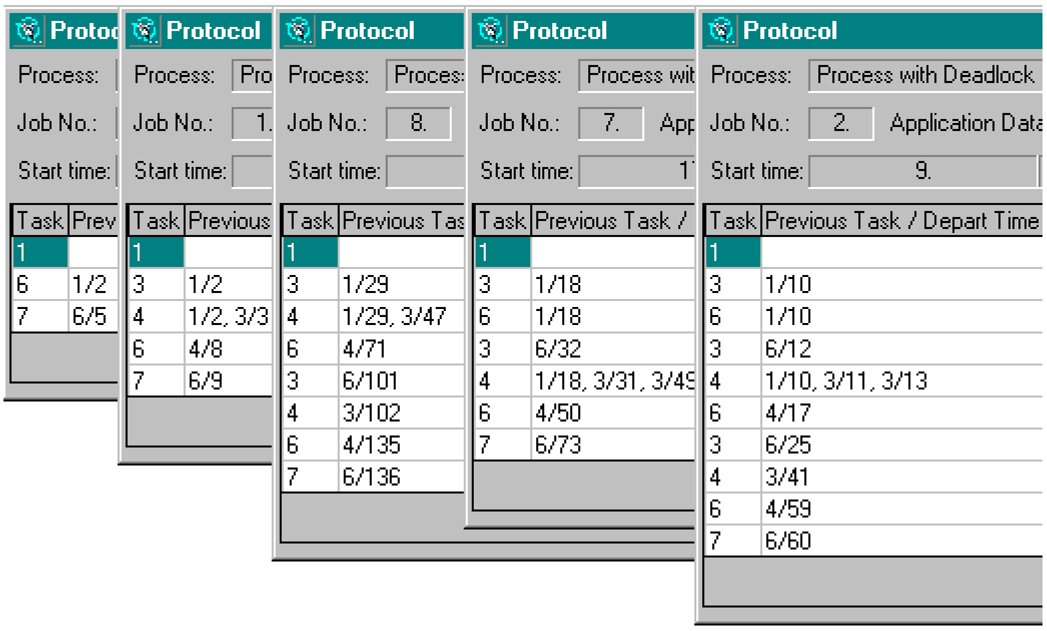
Task	Previous Task / Depart Time
1	
4	1/122
6	4/123, 1/122
3	1/122, 6/124
4	3/139
6	4/159
3	6/183
4	3/184
6	4/185
7	6/186

Figure 6-8. Deadlock Release: 4→6→3

terminate at activity 7 two times: one with execution thread of 1→4→6→7 and another with 1→3→4→6→7.

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



Protoc

Process:

Job No.:

Start time:

Task	Prev
1	
6	1/2
7	6/5

Protocol

Process: Pro

Job No.: 1.

Start time:

Task	Previous
1	
3	1/2
4	1/2, 3/3
6	4/8
7	6/9

Protocol

Process: Proces

Job No.: 8.

Start time:

Task	Previous Tas
1	
3	1/29
4	1/29, 3/47
6	4/71
3	6/101
4	3/102
6	4/135
7	6/136

Protocol

Process: Process wit

Job No.: 7. App

Start time: 1

Task	Previous Task /
1	
3	1/18
6	1/18
3	6/32
4	1/18, 3/31, 3/4
6	4/50
7	6/73

Protocol

Process: Process with Deadlock

Job No.: 2. Application Data

Start time: 9.

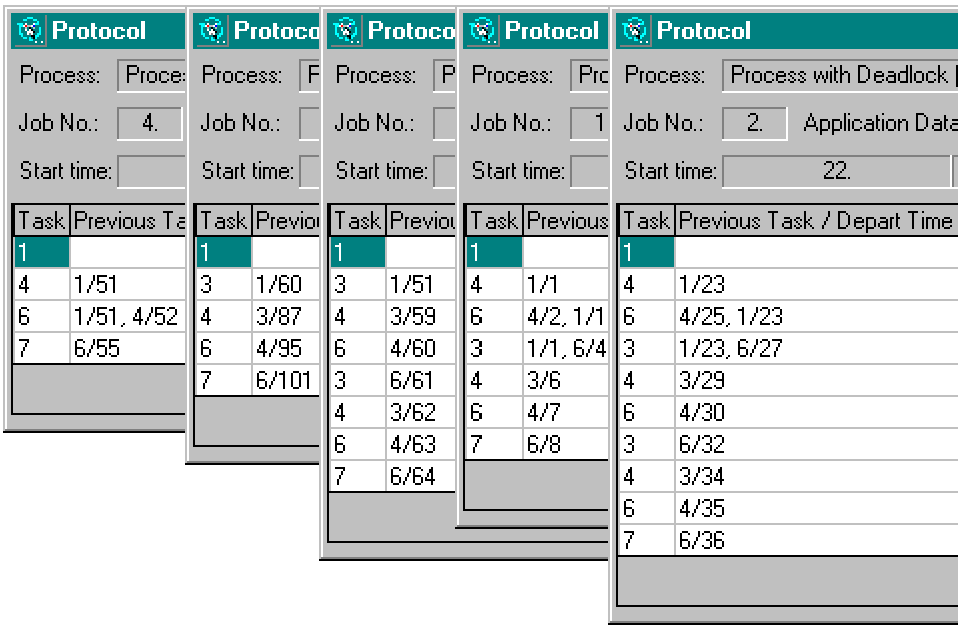
Task	Previous Task / Depart Time
1	
3	1/10
6	1/10
3	6/12
4	1/10, 3/11, 3/13
6	4/17
3	6/25
4	3/41
6	4/59
7	6/60

Figure 6-9. Deadlock Release: 6→3→4

workflow participant of activity 6 choose link "to-end", a process instance will terminate at activity 7 two times.

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.



Process: Proce
Job No.: 4.
Start time:

Task	Previous Ta
1	
4	1/51
6	1/51, 4/52
7	6/55

Process: F
Job No.:
Start time:

Task	Previo
1	
3	1/60
4	3/87
6	4/95
7	6/101

Process: P
Job No.:
Start time:

Task	Previou
1	
3	1/51
4	3/59
6	4/60
3	6/61
4	3/62
6	4/63
7	6/64

Process: Prc
Job No.: 1
Start time:

Task	Previous
1	
4	1/1
6	4/2, 1/1
3	1/1, 6/4
4	3/6
6	4/7
7	6/8

Process: Process with Deadlock |
Job No.: 2. Application Data
Start time: 22.

Task	Previous Task / Depart Time
1	
4	1/23
6	4/25, 1/23
3	1/23, 6/27
4	3/29
6	4/30
3	6/32
4	3/34
6	4/35
7	6/36

Figure 6-10. Deadlock Release: 6→4→3

6.3 Algorithms for Handling Deadlock Situations

The algorithms discussed here are used by the simulator integrated in PM for detecting and releasing deadlock situations arising at run-time. Some of them could also be used by the Espresso workflow engine for the same purposes.

6.3.1 Algorithm for Getting One Deadlock

This algorithm is called by the algorithm for Releasing Deadlock (see Section 6.3.4) during a simulation run. It could also be utilized by the Espresso workflow engine.

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.

Principle

Set U, $|U| > 1$ and $U \subseteq T$, is a parameter of the main procedure. It contains a set of join activities with non-empty ToJoins() (i.e. $\forall j \in U$, ToJoins(j) $\neq \phi$). At each activity in set U, there is a work associated with the same process instance. Work items at some of the join activities in set U may have been involved at run-time in a deadlock situation. If so, this procedure will return such a deadlock cycle of join activities.

A self-called sub procedure with parameters i, k and V is used here for searching a deadlock cycle. 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 $r \in$ ToJoins(q). Current path of join activities built up during searching is kept in stack PathStack(n), n = 1, 2, ..., StackPointer. StackPointer is assigned with 0, before the procedure is called by the main procedure. Join activity j, $j \in$ ToJoins(i), cannot be included in the searching path if $i\#j \notin V$. It is already known that PathStack(1) $\in$ ToJoins(k). Therefore, as a result, the found path of join activities yields a deadlock cycle and will be returned.

Temporary sets RestActivities and CurDeadlock are used in the procedure.

Main Procedure (U)

```
Step 1:  $V \leftarrow \phi$;
Step 2:  RestActivities $\leftarrow U$;
Step 3:  if RestActivities = $\phi$, go to Step 7;
Step 4:  remove an element, say join activity $i$, from set RestActivities;
Step 5:  $\forall j \in$ ToJoins($i$) $\cap U$, let
             $V \leftarrow V \cup \{i\#j\}$
Step 6:  go to Step 3;
Step 7:  if $|V| \leq 1$(join activity pairs in set $V$ cannot yield a deadlock cycle),
         stop (return $\phi$);
Step 8:  remove an element, say join activity pair $i\#j$, from set $V$;
Step 9:  StackPointer $\leftarrow$ 0; CurDeadlock $\leftarrow$ the sub procedure with parameters $j$,
         $i$ and $V$;
Step 10: if CurDeadlock $\neq \phi$, stop (return CurDeadlock);
Step 11: go to Step 7.
```

Sub Procedure (i, k, V)

```
Step 1:  StackPointer $\leftarrow$ StackPointer + 1, PathStack(StackPointer) $\leftarrow i$;
Step 2:  if $i = k$ (cycle of join activities exists), stop (return sequence of join
         activities on the searching path: PathStack(1), PathStack(2), ...,
         PathStack(StackPointer));
```

```
Step 3:  RestActivities ← ToJoins($i$);
Step 4:  if RestActivities = ϕ, go to Step 10;
Step 5:  remove an element, say join activity $j$, from set RestActivities;
Step 6:  if $i\#j \notin V$, go to Step 4;
Step 7:  if $j$ = PathStack($n$) with $n \in [1, \text{StackPointer}]$ (activity $j$ is on current
         searching path), go to Step 4;
Step 8:  CurDeadlock ← the sub procedure self with parameters $j$, $k$ and $V$;
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

Work 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. CopyFlag(*i*) and JoinFlag(*i*), $i \in T$, represent respectively whether activity *i* is a parallel activity and a join activity. ToJoins(*i*) stands for a set of join activities where a work at activity *i* has the potential to be joined.

Principle

WorkID and *i* are parameters of the procedure. WorkID is the identical name of a work in the Espresso WfMS. From work WorkID, it can be known, with which process instance of a process definition the work is associated. This procedure will be called when routing work WorkID to activity *i* of the process definition. Status of work WorkID will be set to "WaitForJoin", if it must wait at activity *i* for joining before it is executed.

Temporary sets ParallelInstances and *U* are used.

Procedure (WorkID, *i*)

```
Step 1:  if JoinFlag($i$) = 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
         work WorkID in set ParallelInstances;
Step 3:  if ParallelInstances = ϕ (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 CurWork, from set ParallelInstances,
         and suppose that it is at activity $k$;
Step 5:  if $k = i$ (works CurWork and WorkID are at the same activity), go to
         Step 11;
Step 6:  if CopyFlag($k$) = FALSE (activity $k$ is not a parallel activity, so work
         CurWork cannot be joined at activity $i$), go to Step 3;
```

```
Step 7:  if $i \notin$ ToJoins($k$) (it is not possible that work CurWork flows from
         activity $k$ to activity $i$), go to Step 3;
Step 8:  set status of work WorkID to “WaitForJoin”;
Step 9:  call Detecting Deadlocks; stop;
Step 10: (work WorkID does not have status “WaitForJoin” and can be sent to
         the workflow participants who will execute activity $i$) stop;
Step 11: join 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

The algorithm is called by the algorithm for Detecting Deadlocks (see Section 6.3.5) during a simulation run. It could also be used by the Espresso workflow engine.

Hypothesis
From the nomination of a work in a WfMS, it can be identified, with which process instance of a process definition it is associated.

Principle
WorkID is a parameter of the procedure. It is known that the work with identical name WorkID has “WaitForJoin” status, which is set in the algorithm for Routing Work to Activity. Suppose that 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*, a deadlock situation may arise at run-time. If so, this 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 |ParallelInstances| = 0 (work WorkID is not parallel anymore), stop
        (return work WorkID);
Step 3: WaitInstances $\leftarrow \phi$;
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 $\phi$);
Step 6: if work CurWork does not have status “WaitForJoin”, stop (return $\phi$);
Step 7: WaitInstances $\leftarrow$ WaitInstances $\cup$ {CurWork};
Step 8: if ParallelInstances $\neq \phi$, go to Step 4;
Step 9: let WaitInstances $\leftarrow$ WaitInstances $\cup$ {WorkID}; stop (return
        WaitInstances).
```

6.3.4 Algorithm for Releasing a Deadlock

This algorithm is called by the algorithm for Detecting Deadlocks (see Section 6.3.5) during a simulation run. It could be used by the Espresso workflow engine.

Hypothesis
Work items in the WfMS have status "WaitForJoin", if they are waiting at join activities for joining.

Principle
Set U is a parameter of the procedure. It contains parallel work items (associated with the same process instance) with status "WaitForJoin". The procedure tries to release a deadlock yielded by some of 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 for joining and some of them may yield a deadlock situation.

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 is yielded by parallel work items in set
        $U$), stop;
Step 7: get the activity from set CurDeadlock, say activity $j$, which has the
        lowest join priority;
Step 8: clear "WaitForJoin" status of the parallel work waiting at activity $j$;
        allocate the work to the worklist of each workflow participant for
        executing activity $j$; stop.
```

6.3.5 Algorithm for Detecting Deadlocks

During a simulation run, this algorithm is called by the algorithm for Routing Work to Activity or is called when a parallel work associated with a process instance is terminated. It might also be called by the Espresso workflow engine in a WfMS at the scheduled times or when some events occur.

Hypothesis
From the identical name of a work item in a WfMS, it can be recognized, with which process instance of a process definition the work is associated. Work items in the WfMS have status "WaitForJoin", if they are waiting at join activities for joining.

Principle
This procedure tries to release deadlocks yielded by parallel work items with status "WaitForJoin".

Temporary sets RestWaits and *U* will be used here.

Procedure
```
Step 1: put all work items that have status "WaitForJoin" in set RestWaits;
Step 2: if RestWaits = $\phi$, stop;
Step 3: remove an element, say work CurWork, from set RestWaits;
Step 4: $U \leftarrow$ Getting Parallel Waiting Work Items (CurWork); if $U = \phi$ (work
        CurWork is not involved in any deadlock situation), go to Step 2;
Step 5: call Releasing a Deadlock with parameter $U$;
Step 6: RestWaits $\leftarrow$ RestWaits $-$ $U$;
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 at run-time wait at different join activities for one another for joining. Thus, a deadlock situation of the process instance arises.

Join priorities are used at run-time to release a deadlock situation. They are assigned in PM by the process designer to the join activities that are involved in a deadlock cycle. For the priority assignment, deadlock cycles with common join activities are grouped together. The assignment of join priorities affects the execution thread of a process instance.

Figure 6-11 presents how deadlock situations are handled upon workflow control data and join activities.

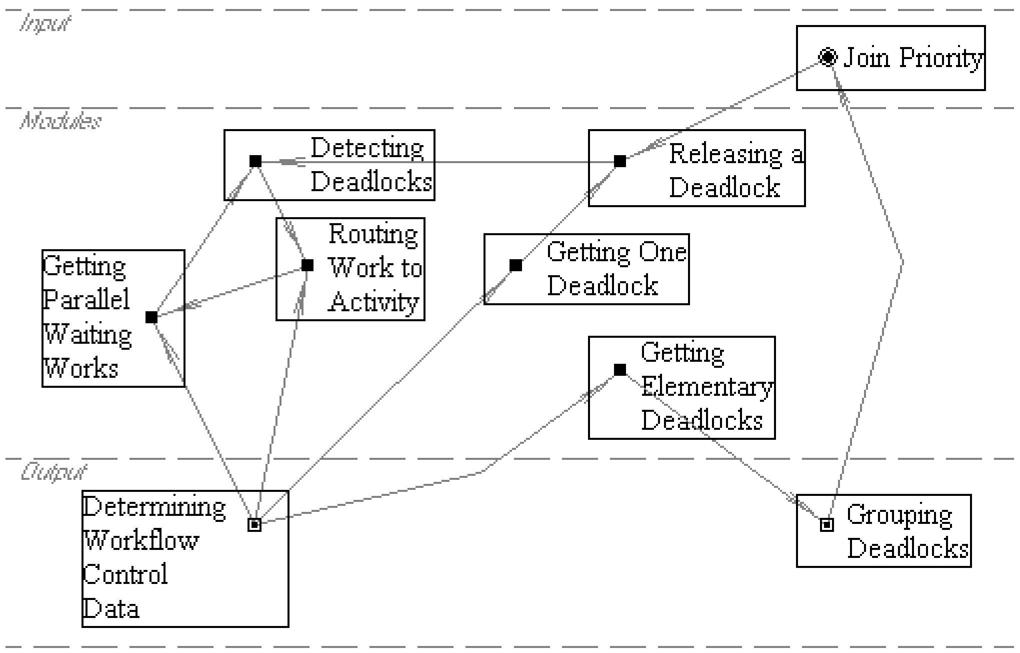


Figure 6-11. Detect and Release Deadlocks

7 SUMMARY

7.1 Definitions and Algorithms

The symbols listed in appendix “Symbols in Definitions and Algorithms” have been used for the process design, verification and simulation.

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

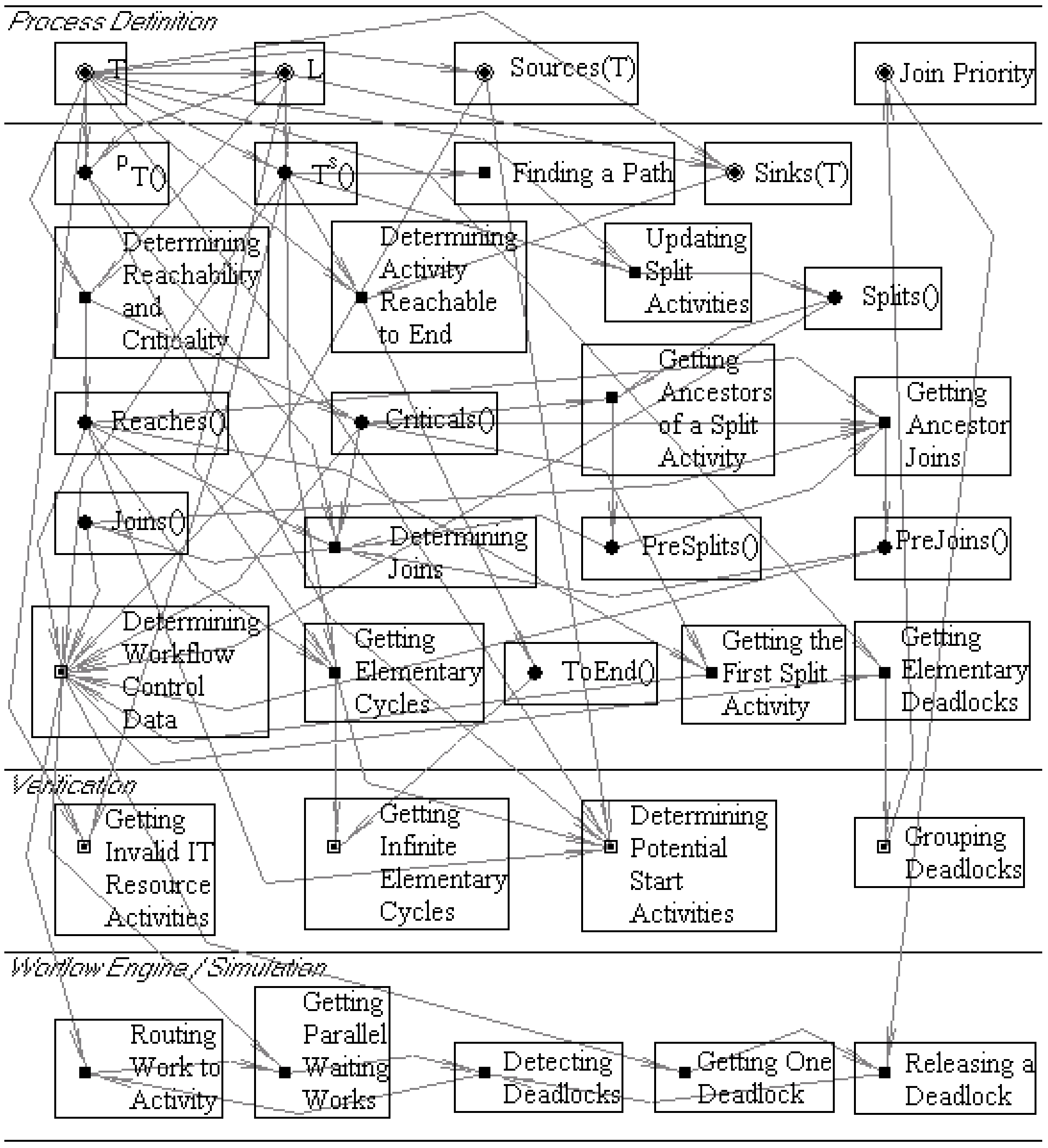

Process Definition
T
L
Sources(T)
Join Priority
^{P}T()
T^{S}()
Finding a Path
Sinks(T)
Determining Reachability and Criticality
Determining Activity Reachable to End
Updating Split Activities
Splits()
Getting Ancestors of a Split Activity
Reaches()
Criticals()
Getting Ancestor Joins
Joins()
Determining Joins
PreSplits()
PreJoins()
Determining Workflow Control Data
Getting Elementary Cycles
ToEnd()
Getting the First Split Activity
Getting Elementary Deadlocks
Verification
Getting Invalid IT Resource Activities
Getting Infinite Elementary Cycles
Determining Potential Start Activities
Grouping Deadlocks
Worflow Engine / Simulation
Routing Work to Activity
Getting Parallel Waiting Works
Detecting Deadlocks
Getting One Deadlock
Releasing a Deadlock

Figure 7-1. Summary of Process Definitions and Algorithms

Process Definition includes the user interfaces where a process designer models a process definition, and assigns join priorities if there are deadlock cycles in the process definition. The Verification area contains the interfaces for outputting verification results. The algorithms in the area of Workflow Engine / Simulation are used by the simulator and could also be used by the workflow engine.

7.2 Constraints in Process Design 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 design and enactment. Therefore, it is quite comfortable and flexible to use PM to design 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.

For the creation of process instances in the Espresso WfMS, start activities of a process definition must be specified. Every activity in a process definition must be reachable from a start activity.

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

If there are deadlock cycles in a process definition, join priorities are used for releasing at run-time a deadlock situation.

Animation and the process protocol reports (see Chapter 13) can help the designer to foresee where parallel work items will be joined and how deadlocks of join activities will be released.

7.3 Further Work*

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

- Search 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. Therefore, work flowing along link (i, j) must 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 no sense and therefore is redundant.

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

- Define synchronization.
  In Figure 7-2, the deadlock situation between activity 4 and activity 5 might be used for 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. Thus, by releasing the deadlock, both the activities can run further without need of waiting for joining.

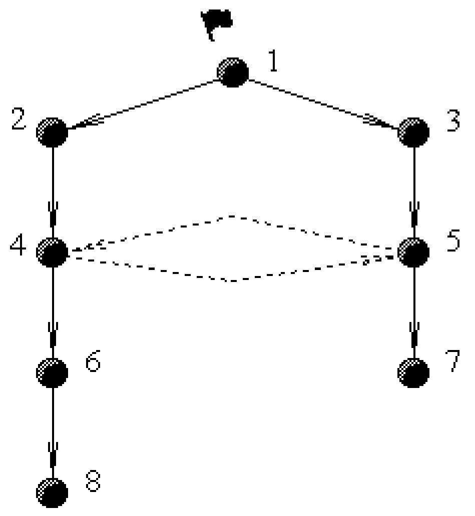


Figure 7-2. Deadlock for Synchronization

- Arrange activities automatically.
  Use the concepts and algorithms of partition and tearing (see [Steward, 1981])

  1) to group the activities on a cycle for automatic simplification of the network of a process definition,
  2) to ungroup the activity clusters that have been so grouped, and
  3) to arrange activities on the process map (e.g. to arrange a specific start activity at most top-left position).

- Suggest a link combination for breaking infinite cycles.
  See [Steward, 1981]. For example, a non-"Always" link that is involved in different cycles the most times could be suggested as a feedback link.

- Determine shortest and longest process duration.
  Shortest duration of a process instance from creation to termination could be determined by supposing that no feedback happens. Slack might also be used for determining the earliest and latest duration of each activity.

Longest duration could be determined according to the specified maximum number of feedbacks of an activity in an escalation definition, or according to the 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 executions time of the activity).

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

Here are the definitions and the algorithms (see [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 specified 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 not are acted on by the system are causes of the relationships or system input variables, and others are effects of the relationships. Therefore, 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/Balintfy /Burdick/Chu, 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. For simulating a WfMS, for example, many model parameters should be estimated or guessed. "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 or the parameters including any of above-mentioned circumstances cannot represent a real world system, and thus no useful conclusion can be drawn from the simulation study upon such a model with those parameters.

"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 /Lieberman, 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/Gnugnoli, 1972, p. 35]

Furthermore, 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 cannot 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/Lieberman, 1974, p. 620].

In addition, simulation brings other advantages. The principal reasons for choosing computer simulation are summarized below (see [Naylor/Balintfy /Burdick/Chu, 1966, pp. 8-9]).

- 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.
- Through simulation one can study the effects of certain informational, organizational, and environmental changes on the operation of a system.
- Detailed behavior observation of the system being simulated may lead to a better understanding of the system and to suggestions for improving.
- Simulation can be used as a pedagogical device for teaching basic skills in theoretical analysis, statistical analysis, and decision-making.
- 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.
- Simulation of complex systems can yield valuable insight into how variables, which represent system attributes, interact.
- Simulation can be used to experiment with new situations about which little or no information is prepared for what may happen.
- Simulation can be used to try out new system operating policies, before running the risk of experimenting on the real system.
- In addition to information about expected values and moments, simulation offers the sequence of events that cause a stochastic system change.
- A dynamic system can be simulated in either real time, compressed time, or expanded time.
- 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.
- 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

Figure 8-1 displays the concepts of the system discussed in this section.

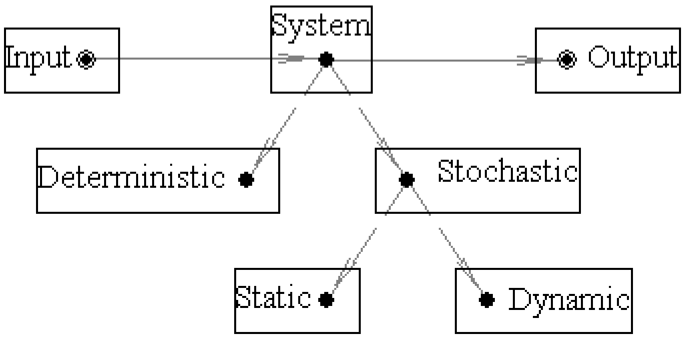


Figure 8-1. System and Categorization

"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/Gnugnoli, 1972, p. 8]

"The objectives 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/Gnugnoli, 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 (e.g. 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 a WfMS is a system that defines, creates and manages the execution of business processes involving people, 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, however, 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 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 (called also a *simulator*) 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/Lieberman, 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/Lieberman, 1974, p. 625]

The Espresso simulation model (the Espresso simulator), discussed in Part Three of this work, is implemented in PM. It is a visual tool for simulating performance or behavior of automated business processes created and routed in the Espresso WfMS in accordance with the process definitions designed with PM. For example, applying the defined join priorities, the simulator releases a deadlock situation of a simulated process instance (see Section 6.2.3). The input of the model is specified in several simulation settings (see chapters 11, 12 and 13) and the output is categorized in diverse simulation reports (see Chapter 13). The cause-and-effect relationships programmed in the simulation model are based upon the definitions and assumptions for process design and simulation. Some of them are described with the algorithms.

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." [Green/Hartley/Maritsas/Powner /Rumsey/Walker, 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 work item to the worklist of a workflow participant causes him to become busy when he is idle; a completion of a work causes the workflow participant to become idle when no other work items are waiting in his worklist. The events in a WfMS are discrete, because they are countable.

"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]

There are three things needed to perform the discrete-event simulation (see [Lewis/Smith, 1979, p. 104]).

- *Clock*: keeps track of the simulated time. It is updated by the model algorithm and initialized with a zero value.
- *Eventlist*: a list of scheduled events that are 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, different events will be scheduled during a simulation run. For each kind of scheduled event, there is an event-processing algorithm. The eventlist is usually sorted to the nearest scheduled time for picking out the next occurring event.

The simulator integrated in PM uses the next-event approach as the model algorithm. The unit of the clock could be decided by the system analyst before a simulation run. It can be minute, hour, day or week. When unifying a time from a larger unit (such as week or day) to a smaller unit (such as hour or minute), the standard weekly working hours are considered and weekends (Saturday and Sunday) as well as holidays are excluded. The *standard weekly working hours* (e.g. 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

The next-event approach is implemented in the simulator integrated in PM as the model algorithm.

Hypothesis
SimulationClock represents the clock in the simulator. 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 initialized to the system states when a simulation run begins. During the simulation run, the clock is advanced to the occurrence time of the first events kept in the eventlist, and hence the event occurs. During processing 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. The simulation run will be terminated when the clock exceeds a specified simulation end time, when there is no event in the eventlist anymore, or when the system analyst interrupts the simulation run.

Procedure
```
Step 1:  get input data from different simulation settings;
Step 2:  clear all statistic variables; initialize all state variables;
Step 3:  SimulationClock ← 0;
Step 4:  initialize the eventlist and ensure that it is not empty (e.g. for
         simulating a WfMS, schedule a process creation event for each start
         activity of an analyzed process definition);
Step 5:  remove the first (the nearest occurring) event from the eventlist and
         suppose that it occurs at time t;
Step 6:  SimulationClock ← t; if the value of SimulationClock exceeds the
         specified simulation end time, go to Step 10;
Step 7:  if the event is to terminate the 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 sorted to the times);
Step 9:  if the eventlist is not empty, go to Step 5;
Step 10: generate simulation reports from the statistic variables; stop.
```

8.2.2.3 Data Structure of the Eventlist

In PM, data of a scheduled event are kept in an event record. The Eventlist containing the event records is sorted in respect to event occurrence time in a

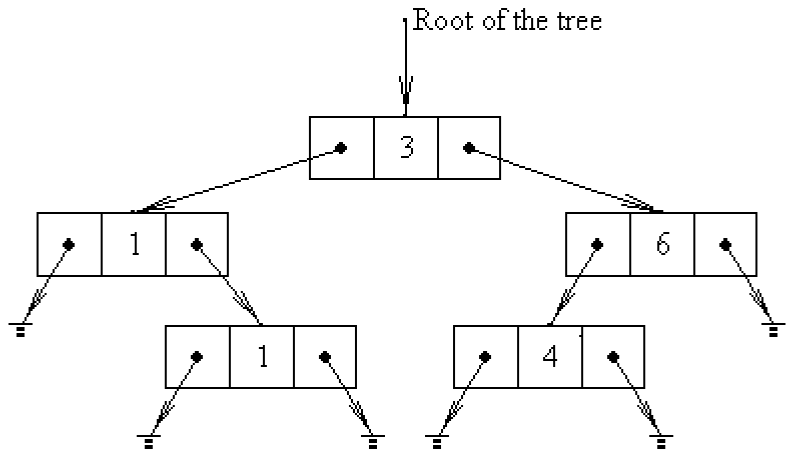


Figure 8-2. Data Structure of a Binary Tree

binary tree structure (see Figure 8-2) for dynamically inserting a new created event record into the eventlist or removing the event record of the first occurring event from the eventlist. Each event has two pointers: a left pointer and a right pointer. The left pointer connects to an event that occurs earlier and the right pointer to one that occurs later.

8.2.2.3.1 Algorithm for Inserting an Event Record into the Eventlist

This procedure is called when a new event is scheduled. The record keeping the event is inserted into the eventlist sorted in respect to the occurrence time.

Hypothesis
RootEventlist is the pointer connecting to the root of the tree structure of the eventlist. OccurTime(*i*), Previous(*i*) and Next(*i*) represent respectively the occurrence time, the left pointer and the right pointer of the event kept in record *i*. A pointer has value 0 if it connects to nothing.

Principle
Here RootTree, CurRecord, *i* and IsPriori are the parameters of the procedure. The event kept in record *i* will be inserted into the tree in respect to the occurrence time. RootTree represents the root pointer of the tree. CurRecord is the current treated record in the tree. IsPriori means whether record *i* should be inserted before any other records with the same occurrence time. This procedure will be called by the next-event approach with parameters RootEventlist, RootEventlist, *i* and FALSE (or TRUE), when a new event kept in record *i* is scheduled.

```
Procedure (RootTree, CurRecord, i, IsPriori)
Step 1: if CurRecord ≠ 0, go to Step 3;
Step 2: (the eventlist is empty)
        1° RootTree ← i;
        2° Previous(i) ← 0 and Next(i) ← 0;
        3° stop;
Step 3: if OccurTime(i) < OccurTime(CurRecord), go to Step 6;
Step 4: if OccurTime(i) = OccurTime(CurRecord) and IsPriori = TRUE, go to
        Step 6;
Step 5: (record i should be inserted after record CurRecord) if Next(CurRecord)
        = 0 (the record i can be directly connected by the right pointer of record
        CurRecord),
            1° Next(CurRecord) ← i;
            2° Previous(i) ← 0 and Next(i) ← 0;
            3° stop;
        otherwise, call the procedure self with parameters RootTree,
        Next(CurRecord), i and IsPriori and stop;
Step 6: (record i should be inserted before record CurRecord) if
        Previous(CurRecord) = 0 (the record i can be directly connected by the
        left pointer of record CurRecord),
            1° Previous(CurRecord) ← i;
            2° Previous(i) ← 0 and Next(i) ← 0;
            3° stop;
        otherwise, call the procedure self with parameters RootTree,
        Previous(CurRecord), i and IsPriori and stop.
```

8.2.2.3.2 Algorithm for Removing the First Event from the Eventlist

This procedure is called when the simulation clock can be advanced. The event record with the earliest occurrence time will be removed from the eventlist. Consequently, the clock can be advanced to this time and so the event occurs.

Hypothesis
RootEventlist is the pointer connecting to the tree root of the eventlist sorted in respect to the occurrence time. Previous(*i*) and Next(*i*) represent respectively the left pointer and the right pointer of the event kept in record *i*. A pointer has value 0 if it connecs to nothing.

Principle
This procedure returns the first record in the tree structure. Here RootTree, CurRecord and FatherRecord are the parameters of the procedure. RootTree represents the root pointer of the tree. Record CurRecord in the eventlist is the

current treated record and is connected by the left pointer of record FatherRecord. This procedure will be called by the next-event approach with parameters RootEventlist, RootEventlist and 0, when the simulation clock can be advanced.

```
Procedure (RootTree, CurRecord, FatherRecord)
Step 1: if Previous(CurRecord) ≠ 0 (record CurRecord is not the first record in
            the tree), i ← call the procedure self with parameters RootTree,
            Previous(CurRecord) and CurRecord and go to Step 4;
Step 2: (record CurRecord is the first record in the tree) let i ← CurRecord;
Step 3: (to disconnect record i from the tree) if FatherRecord = 0 (record
            CurRecord is connected by the root pointer of the tree), let
                  RootTree ← Next(CurRecord);
            otherwise (to change connection of the left pointer from record
            FatherRecord) let
                  Previous(FatherRecord) ← Next(CurRecord);
Step 4: stop (return i).
```

8.2.3 Variables

System factors and attributes are represented in a simulation model by variables. Variables used in a simulation model can be classified as decision variables, system parameters, state variables, statistic variables, or performance criteria. Figure 8-3 illustrates the role of the simulation model in providing a description of system behavior.

Decision variables and system parameters are input variables of the simulation model and are assumed to have been predetermined in different simulation settings before a simulation run in PM. Some of these input variables are, for all practical purposes, deterministic, whereas others are stochastic.

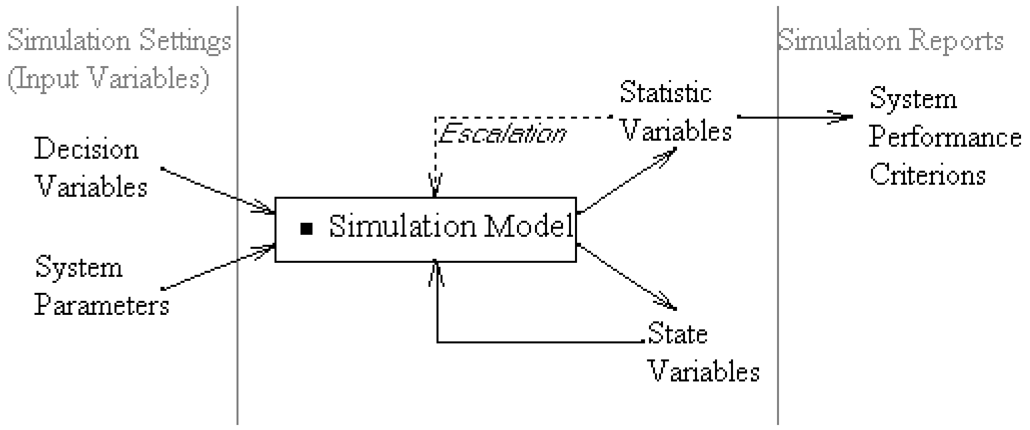


Figure 8-3. Simulation Model and Variables

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

8.2.3.1 Decision Variables

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 variable*s.

The values chosen for the decision variables will affect the state of the system or system performance. Therefore, 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 could be

- different process definitions implemented in a WfMS;
- environment design and specification of the WfMS, consisting of relevant application databases as well as the organizational model;
- life period specification of a process definition;
- network structure of a process definition (i.e. the sets of activities, links and start activities);
- human and material resources assigned to each activity;
- resource availability (definitions of organizational roles in the PAVONE Organization Databases and/or the Notes Organization Directories, including specification of weekly work hours of a resource for a process definition);
- queuing rule (e.g. first-in-first-out) for work items in a workflow participant's worklist;
- escalation conditions.

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.

"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 Parameters

*System parameter*s 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.

Every system parameter of a simulation model influences a state variable or a statistic variable. Therefore, the state variables and statistic variables are dependent upon system parameters too.

Here are some system parameters appearing in the Espresso WfMS:

- fix costs for execution of an activity;
- resource hourly costs;
- the time interval between two conjunctive created process instances, called *intercreation time* of process instances, at each start activity of a process definition;
- probability for routing work along a "Multiple Choice" link;
- routing distribution among "Exclusive Choice" outgoing links of an activity;
- value distribution 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 Variables

The *state* of a system can be thought of as the totality of all relevant system characteristics. It varies as time elapses. *State variable*s describe the state of a system or one of the system 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 affected by some statistic variables that are used in an escalation condition. An invoked escalation can change the system state.

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

The state variables in the Espresso WfMS include

- the clock for tracking simulation time,
- status of a workflow participant (i.e. busy or idle),
- the number of running process instances in the WfMS,
- the state of a running process instance (being 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 Variables

Statistic variables are accumulated data from state variables as well as input variables. They updated during a simulation run according to the statistical 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: total number of created process instances in accordance with a process definition, for example. This kind of statistic variable can be directly used as a system performance criterion.
- Accumulation of a state variable: total activity execution time, for example. It can be used for computing an average data (here, the average activity execution time), through dividing the sum by the count of accumulated data (here, total number of completed activities).
- integral of a state variable evaluated within a time range. It is prepared for calculating time average data, such as average queue length during a simulation run.

The time average data of a variable is calculated through dividing the integral of the variable evaluated within a time range by the length of the time range. For example, the plot in Figure 8-4 illustrates the values of the state

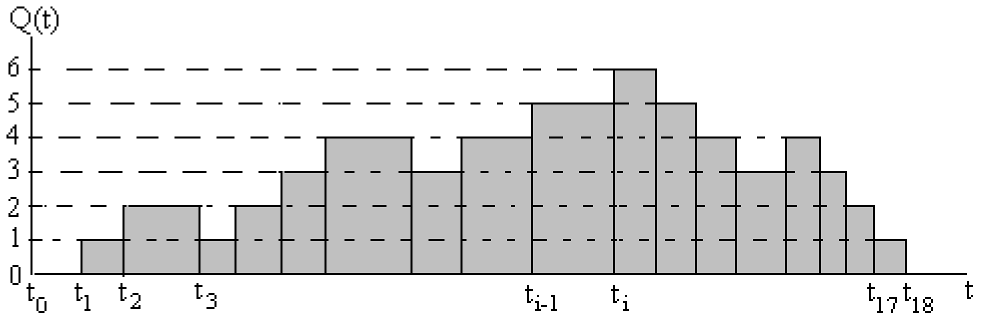


Figure 8-4. Calculation of the Integral Evaluated within a Time Range

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} is calculated by (see [Tijms, 1986, p. 15])

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

The integral of $Q(t)$ evaluated within range $[t_0, t_{18}]$ in the formula is in fact the shaded area in Figure 8-4. Because the value of a state variable is changed only when an event occurs, the integral of a state variable evaluated within range $[t_0, t_{18}]$ 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 (i.e. the queue length just before an event occurs at time t_i). For example, $Q(t_{3-}) = 2$ in Figure 8-4. Therefore, the integral of the queue length evaluated within range $[t_0, t_{18}]$ can be accumulated with value

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

at each time t_i, $i = 1, 2, \ldots, 18$.

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 making decision or choosing policy.

The system performance criteria created by the Espresso simulation model are contained in several simulation statistical 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 rate 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 method for analyzing and experimenting a complex system, such as a WfMS, which is usually a dynamic stochastic system.

A simulator on a digital computer is a programmed model comprised of cause-and-effect relationships of a system. It permits the decision maker to observe the state of a system over time as well as at particular points in time. The 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 seek a satisfactory operating policy, relative to some particular criteria of system performance.

Whether a simulation model is suitable to simulate a real world system or not is determined by the following factors:

- well-chosen pseudorandom number generators,
- appropriate approximate random variate generation techniques,
- no programming errors,
- no model misspecification of cause-and-effect relationships,
- no numerical calculation errors,
- easy usage of the simulation model,
- acceptable expense of computer time for simulation, and
- 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 system analysts should work together to collect data and determine the appropriate input to the simulation model. They are also required for statistically analyzing the output from simulation runs for making decisions—choose the best of the simulated alternative operating policies for the system.

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/Balintfy/Burdick/Chu, 1966, p. 43] The sample space is also called *population*. It is the set of individuals, items, or data from which a statistical sample is taken.

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/Pooch, 1980, p. 23]

Random variables are denoted by capital letters, and random variates by lower case letters. 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, i.e. $Prob(X \leq x)$. In a similar manner, $f(x)$ represents the value of the probability *density function* of the continuous random variable X when $X = x$, while p_i represents the probability of a discrete random variable X when $X = x_i$. See [Naylor/Balintfy/Burdick/Chu, 1966, p. 43]. That is,

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

or

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

Mean, denoted by μ, 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 σ^2, is used as a measure of dispersion. The square root of the variance, that is σ, is called the *standard deviation*. Variance is given by

$$\int_{-\infty}^{\infty} (x-\mu)^2 f(x)dx$$

or

$$\sum_{i=1}^{\infty} (x_i - \mu)^2 p_i$$

See [Fishman, 1973] and [Graybeal/Pooch, 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 the Espresso WfMS are introduced here. The uniform, normal, exponential, gamma or empirical distribution can be specified for generating time and other random variates in the simulation model. The *t*-, χ^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 $U(a, b)$, called also 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

$$f(x) = \begin{cases} 1 / (b - a), & a \le x \le b \\ 0, & otherwise \end{cases}$$

The cumulative distribution function is given by

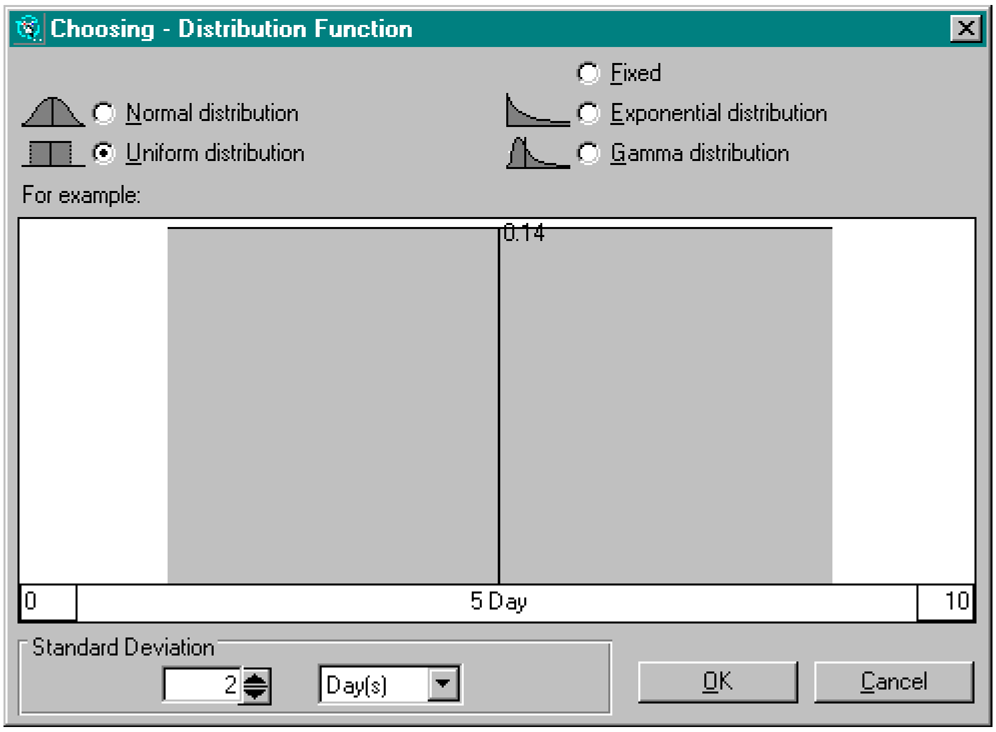
$$F(x) = \begin{cases} 0, & x < a \\ \dfrac{x - a}{b - a}, & a \le x \le b \\ 1, & x > b \end{cases}$$

The mean μ and variance σ^2 for this distribution are (see [Graybeal/Pooch, 1980, p. 47])

$\mu = (a + b)/2$ 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/Pooch, 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." [Green/Hartley/Maritsas/Powner/Rumsey/Walker, 1975, p. 65]



Fixed [✓ Uniform distribution]; Standard Deviation: 2 Day(s)

Figure 9-1. Uniform Distribution with that μ = 5 and σ = 2

An example of the uniform distribution with μ = 5 and σ = 2 is plotted in Figure 9-1. The range of the distribution of this example can be obtained by

$$\mu \pm \sqrt{3}\sigma$$

That is,

$$a = 1.54 \text{ and } b = 8.46$$

In the range [1.54, 8.46], the density function has the constant value of 0.14.

In the Espresso simulation model, it is possible that σ = 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 μ.

A uniform distribution with the range (0, 1) is called *standard uniform distribution*, denoted by $U(0, 1)$. Suppose that the variate u of a random variable U follows the standard uniform distribution, the cumulative distribution function is then given by

$$\text{Prob}(U \le u) = F(u) = (u - a)/(b - a) = u, \text{ for } 0 \le u \le 1$$

That is,

$$\text{Prob}(U \le u) = u, \text{ for } 0 \le u \le 1$$

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/Pooch, 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

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

where μ is the mean and σ 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

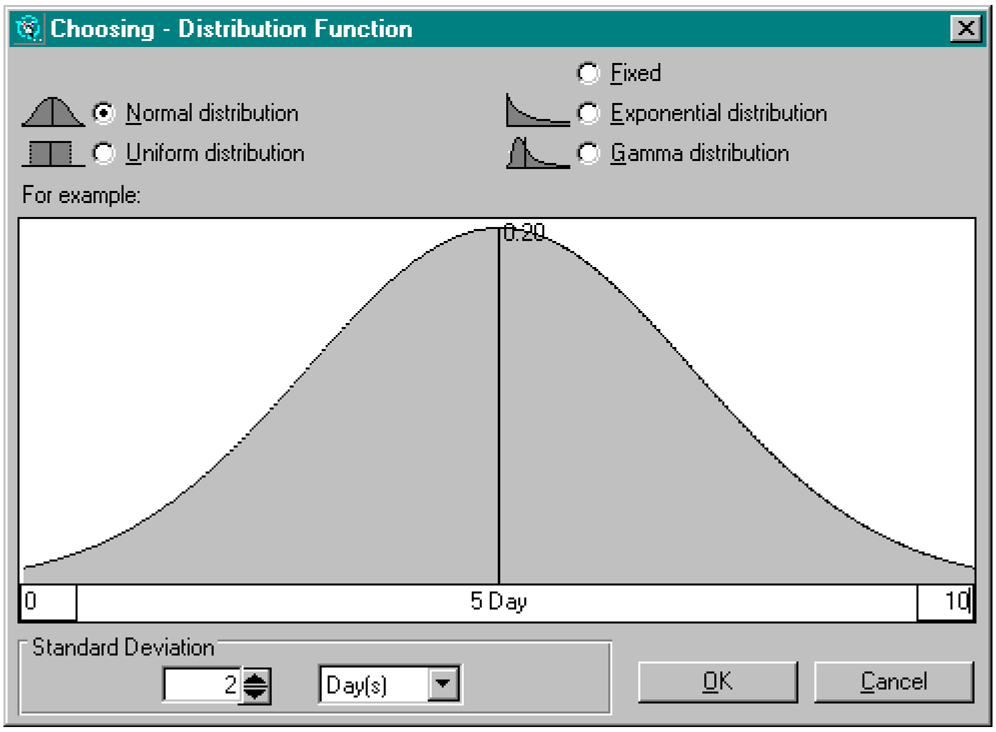


Figure 9-2. Normal Distribution with that $\mu = 5$ and $\sigma = 2$

infinity, where μ and σ^2 are respectively the mean and the variance of X_i ($i = 1, 2, \ldots, n$). See [Graybeal/Pooch, 1980, p. 49 and p. 92]. That is, statistic

$$\frac{\overline{X} - \mu}{\sigma / \sqrt{n}}$$

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/Gnugnoli, 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/Pooch, 1980, p. 49]

The probability density of an exponential distribution is

$$f(x) = \alpha e^{-\alpha x}, x \geq 0$$

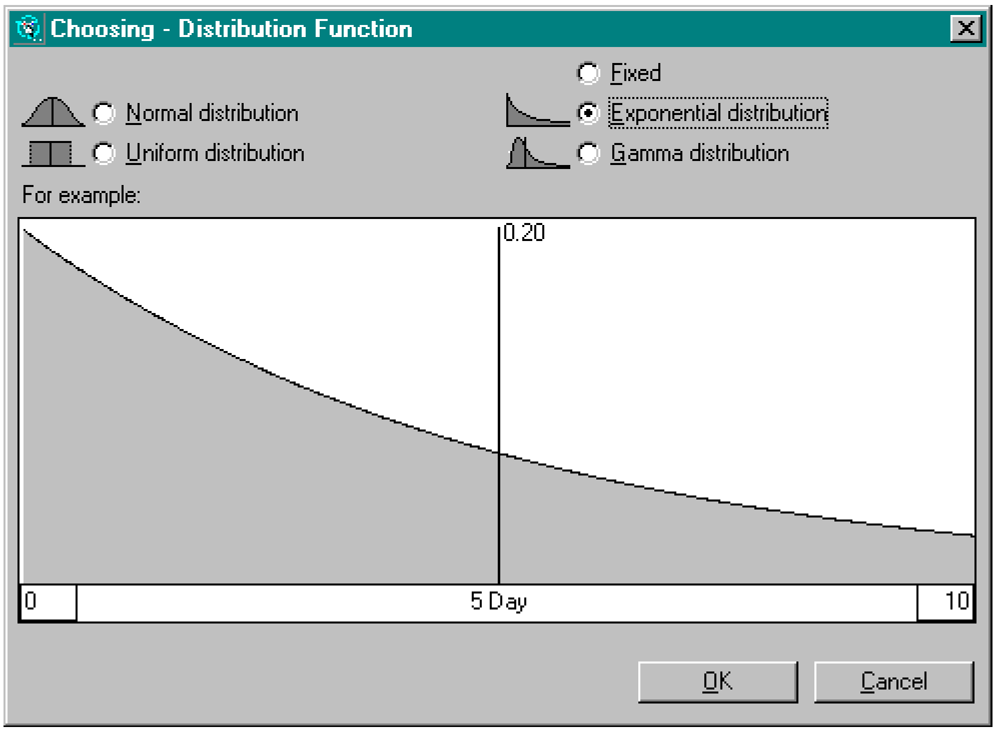
where α is a positive constant. The cumulative distribution function is

$$F(x) = 1 - e^{-\alpha x}$$

The mean μ and variance σ^2 for the exponential distribution are (see [Graybeal/Pooch, 1980, p. 49])

$$\mu = 1/\alpha \text{ and } \sigma^2 = 1/\alpha^2$$

That is, the exponential distribution's mean μ and standard deviation σ are the same.


Choosing - Distribution Function

( ) Normal distribution
( ) Uniform distribution
( ) Fixed
[✓ Exponential distribution]
( ) Gamma distribution

For example:

Figure 9-3. Exponential Distribution with that $\mu = 5$

An example of the density function of the exponential distribution with $\mu = 5$ is illustrated in Figure 9-3.

See [Gottfried, 1984, p. 85] for better comprehension of the physical significance of exponential distribution.

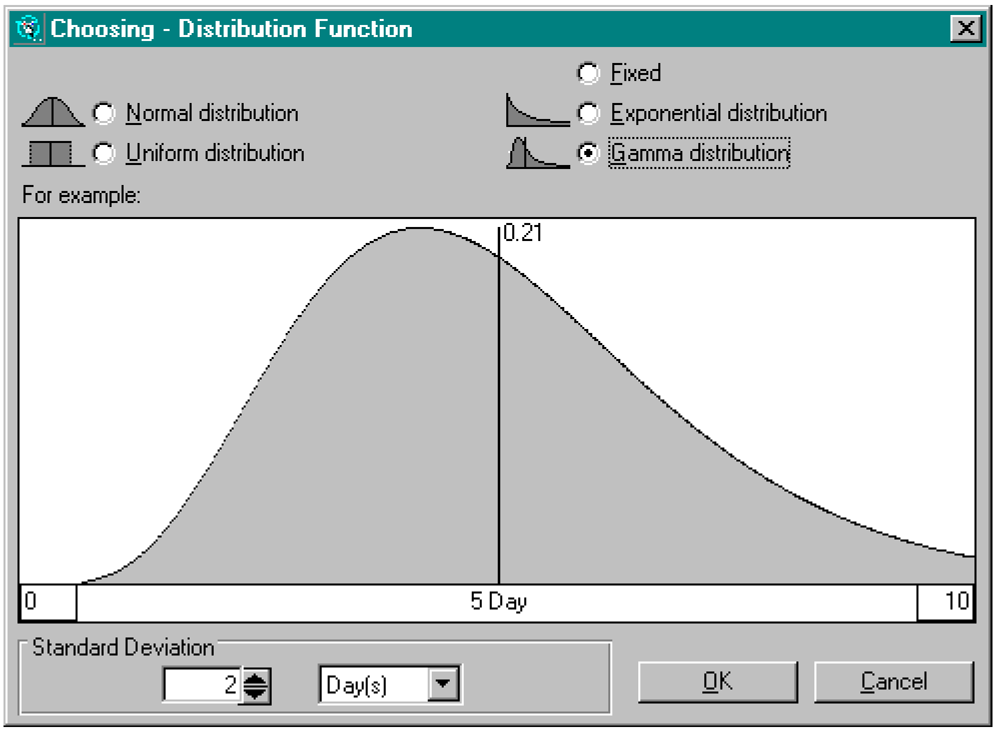
9.2.4 Gamma Distribution

The probability density of the *gamma distribution* is (see [Gottfried, 1984, p. 90])

$$f(x) = \frac{\alpha^{\beta} x^{(\beta-1)} e^{-\alpha x}}{(\beta-1)!}$$

where α is a positive constant and β 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 that $\mu = 5$ and $\sigma = 2$ is given in Figure 9-4.


Choosing - Distribution Function

Normal distribution [✓ Gamma distribution] Uniform distribution Fixed Exponential distribution

Standard Deviation: 2 Day(s)

Figure 9-4. Gamma Distribution with that $\mu = 5$ and $\sigma = 2$

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 β 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 (or $\beta = 1$), the gamma distribution has the plot of an exponential distribution.

The gamma distribution can also be defined for non-integer values of β, although physical applications of this nature are less common (see [Gottfried, 1984, p. 92]). For further discussion see [Fishman, 1973, pp. 208-209].

9.2.4.1 Algorithm for Drawing Gamma Distribution

This algorithm is used by the Espresso simulation model integrated in PM for drawing a plot of a gamma distribution chosen for a random variable (see Figure 9-4) with known mean and standard deviation.

Hypothesis

The parameter β in the probability density function of the gamma distribution is integer-valued.

Principle

Parameters μ and σ are mean and standard deviation of the gamma distribution. As $\mu = \beta/\alpha$ and $\sigma^2 = \beta/\alpha^2 = \mu/\alpha$, so $\beta = \mathrm{INT}(\mu^2/\sigma^2)$ and $\alpha = \beta/\mu$.

For drawing a gamma distribution, $f(x)$ should be evaluated for every given value of x. Directly using the density function may cause the computer to overflow. Thus, it will be calculated via the formula

$$\begin{aligned}\ln f(x) &= \beta \ln \alpha + (\beta - 1) \ln x - \alpha x - \ln[(1)\,(2) \ldots (\beta - 1)] \\ &= \ln \alpha - \alpha x + (\beta - 1)(\ln \alpha + \ln x) - [\ln 1 + \ln 2 + \ldots + \ln(\beta - 1)]\end{aligned}$$

Temporary variable p is used to keep the calculation of $\ln f(x)$.

Procedure (μ, σ)

```
Step 1: let $\beta \leftarrow \mathrm{INT}(\mu^2/\sigma^2)$;
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.

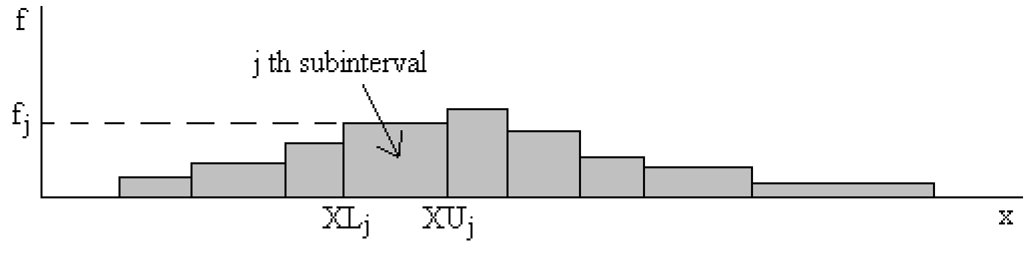


Figure 9-5. Empirical Density Function

There are several adjacent subintervals, which are numbered consecutively (i.e. j = 1, 2, ..., m). Each subinterval is represented as a rectangle with lower and upper interval bounds XL_j and XU_j respectively. The height of each subinterval, denoted by f_j, represents the probability that the value of random variable X will fall into the jth subinterval [XL_j, XU_j]. Since f_j, j = 1, 2, …, m, represents the probability, the sum must equal 1. That is,

$$f_1 + f_2 + \dots + f_m = 1$$

Figure 9-6 shows the cumulative distribution of the distribution presented in

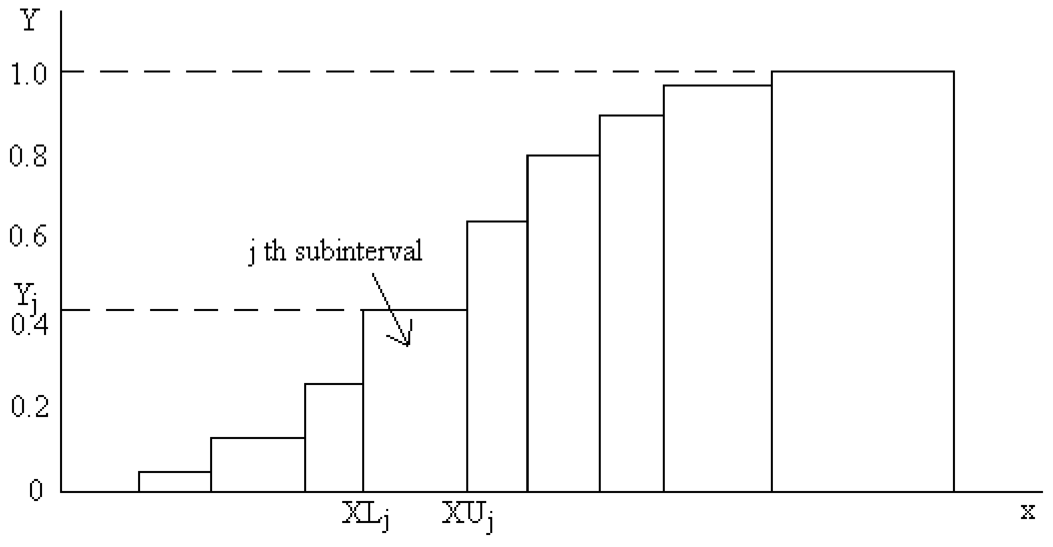


Figure 9-6. Empirical Cumulative Distribution Function

Figure 9-5. The height of each subinterval, Y_j now represents the probability that the value for the random variable X does not exceed XU_j, or

$Y_j = \text{Prob}(X \le XU_j)$

A *0-1 distribution*, or Bernoulli distribution (see [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, say 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 is (see [Maisel/Gnugnoli, 1972, p. 59])

$$f(x) = \frac{\Gamma\left(\frac{n+1}{2}\right)}{\sqrt{n\pi}\,\Gamma\left(\frac{n}{2}\right)\left(1 + \frac{x^2}{n}\right)^{(n+1)/2}}, \textit{for} -\infty < x < \infty$$

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

$$\Gamma(1/2) = \sqrt{\pi} = 1.77245385\ldots\ldots$$

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 (source:

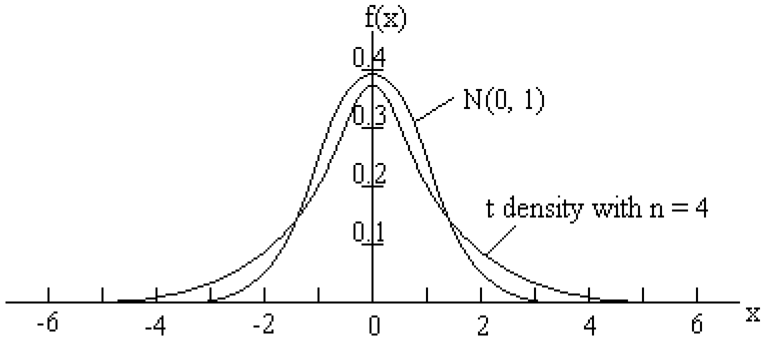


Figure 9-7. t-Distribution and $N(0, 1)$ Distribution

[Maisel/Gnugnoli, 1972, p. 59]). A standardized normal distribution $N(0, 1)$ is plotted on the same coordinates to illustrate the similarity of these distributions.

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 χ^2 distribution with v degrees of freedom. Then $t = Z/(C/v)$ follows a t-distribution with v degrees of freedom. See [Graybeal/Pooch, 1980, p. 51].

The t-distribution arises quite often when normal distributions are sampled. If $\overline{Y}$ denotes the sample mean, s the sample standard deviation, and n the size of the sample, the statistic

$$t = n^{1/2}(\overline{Y} - \mu)/s$$

follows a t-distribution with $n - 1$ degrees of freedom. See [Graybeal/Pooch, 1980, p. 51].

9.2.7 χ^2 Distribution*

The χ^2 (*chi-square*) *distribution* is used in goodness-of-fit tests and in certain nonparametric test. Its density function is

$$f(x) = \frac{x^{(n/2)-1}e^{-x/2}}{\Gamma(n/2)2^{n/2}}, \; for\ 0 \le x < \infty$$

Here n is the degrees of freedom. For the χ^2 distribution, the mean is n, and the variance is $2n$.

χ^2 density functions for several values of n are plotted in the Figure 9-8 (source: [Maisel/Gnugnoli, 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

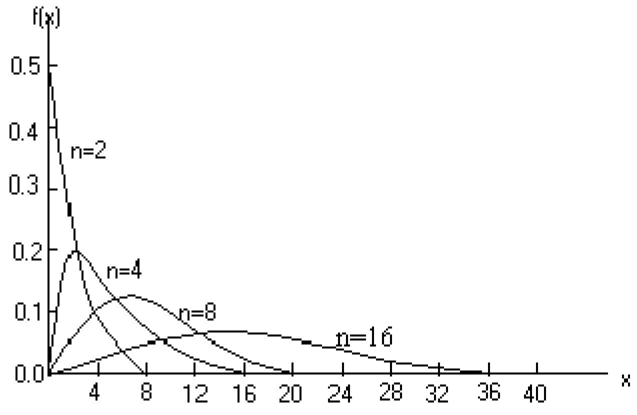


Figure 9-8. χ^2 Distribution with Different Freedom n

random variables, then $Z_1^2 + Z_2^2 + ... + Z_n^2$ is a χ^2 distribution with n degrees of freedom. See [Graybeal/Pooch, 1980, p. 51].

9.2.8 *F*-Distribution*

The *F-distribution* is used primarily to test ratios of variances between two samples from normally distributed populations. Its density function has two parameters, one associated with each of the two variances involved in forming the *F*-ratio. Using m to denote the degrees of freedom associated with the variance in the numerator of the ratio and n to denote the degrees of freedom associated with the variance in the denominator, the density function is (see [Maisel/Gnugnoli, 1972, p. 60])

$$f(x) = \frac{\Gamma[(m+n)/2]}{\Gamma(m/2)\Gamma(n/2)} m^{m/2} n^{n/2} x^{(m/2)-1} (m+nx)^{-(m+n)/2}, \quad for \quad 0 \le x < \infty$$

The mean of the *F*-distribution is

$$m/(n-2)$$

and the variance is

$$[m^2(n+2)]/[m(n-2)(n-4)]$$

The *F*-distribution is also useful in hypothesis testing. Let C_1 and C_2 be two independent χ^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/Pooch, 1980, p. 51].

This distribution arises when one is sampling from two normal populations. Let sample 1 consists of n_1 points from a normal population with mean μ_1 and variance σ_1^2, and let s_1^2 be the sample variance. Let sample 2 consists of n_2 points from a normal population with mean μ_2 and variance σ_2^2, and let s_2^2 be the sample variance. Then $(s_1^2/\sigma_1^2)/(s_2^2/\sigma_2^2)$ is distributed according to the *F*-distribution with $n_1 - 1$ and $n_2 - 1$ degrees of freedom. See [Graybeal/Pooch, 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/Pooch, 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/Pooch, 1980, pp. 59-60]

Suppose that 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 μ is estimated by

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Variance σ^2 is estimated by

$$s^2 = \frac{1}{n}\sum_{i=1}^{n} (x_i - \bar{x})^2$$

or equivalently, as

$$s^2 = (\frac{1}{n}\sum_{i=1}^{n} x_i^{\,2}) - (\bar{x}^2)$$

Standard deviation σ is then estimated by s.

Note that the equations for estimating σ^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, there are two general approaches. "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/Pooch, 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/Gnugnoli, 1972, p. 45]

Suppose that x_i, $i = 1, 2, \ldots, n$, represents the ith value in a sample of a random variable X. Sometimes the observed (or computed) values of x_i's are grouped into successive intervals. A set of relative frequencies, which represents the density function of an empirical distribution, can be obtained from these grouped data. Let n_j be the number of values of x_i 's falling into the jth interval (where j is an interval index that ranges from 1 to m), i.e.

$$n = \sum_{j=1}^{m} n_j$$

Then the relative frequency for the jth interval f_j is

$$f_j = n_j/n$$

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

$$\begin{cases} Y_1 = f_1 \\ Y_2 = f_1 + f_2 \\ \dots \\ Y_j = f_1 + f_2 + \dots + f_j \\ \dots \\ Y_m = \sum_{j=1}^{m} f_j = 1 \end{cases}$$

Here

$$Y_j = \text{Prob}(X \le XU_j)$$

with that XU_j is the upper bound of jth 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 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/Gnugnoli, 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 α, is the probability that a valid hypothesis will be rejected. The quantity 1 – α is called the *level of significance of the test*. Specifications commonly are designed to ensure that α ≤ 0.05, but this is an arbitrary value, and larger type I errors might be accepted in particular situations. See [Maisel/Gnugnoli, 1972, p. 68].

The size of the type II error, denoted by β, is related to the degree of falsity of the hypothesis being tested. For example, suppose that in a 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 – β is called the *power of the test.*

"Obviously one of the objectives in hypothesis testing is to minimize α and β, 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 α is set as some predetermined acceptable level and the decision rule is formulated to minimize β." [Graybeal/Pooch, 1980, p. 62]

To compute β, one must assume the hypothesis, e.g. "μ = 12", is false and another alternate hypothesis, e.g. "μ = 14", is true. For a given statistical test (with fixed sample size and fixed α), β and the power (1 – β) can then be calculated corresponding to various assumed levels for the extent of falsity of the hypothesis as shown in Figure 9-9 (source: [Graybeal/Pooch, 1980, p. 65]).

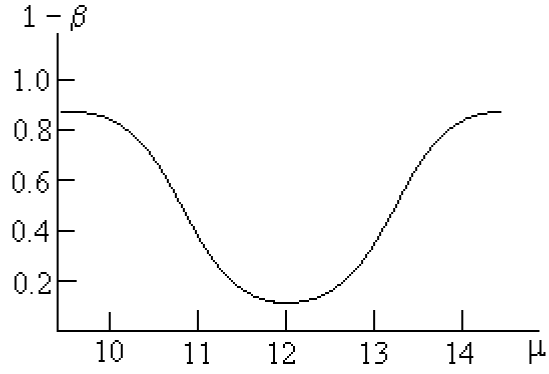


Figure 9-9. A Power Curve

In the following sections, it will be outlined how to test a hypothesis with given α. In a business environment, α is called a *rejection probability* and (1 – α) a *confidence level*. The interval, within which random variable *X* falls, is determined via the statistical test at a given confidence level and hence is called *confidence interval*.

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

$$(\frac{\overline{Y}-\mu}{s})\sqrt{n-1}$$

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

$$-t_{n-1,1-\alpha/2} < (\frac{\overline{Y}-\mu}{s})\sqrt{n-1} < t_{n-1,1-\alpha/2}$$

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 α/2 represents either of the shaded areas shown in Figure 9-10 (see [Gottfried, 1984, p. 168]). Note that the

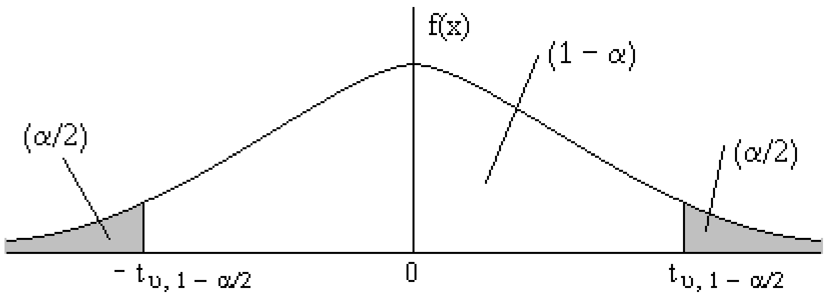


Figure 9-10. The Critical Region of *t*-Distribution

quantity (1 – α) is the corresponding confidence level, that is, the likelihood that the value obtained from equation

$$(\frac{\overline{Y}-\mu}{s})\sqrt{n-1}$$

will fall within the unshaded area in Figure 9-10.

It is more convenient to rewrite the equation as

$$\overline{Y} - t_{n-1,1-\alpha/2} s / \sqrt{n-1} < \mu < \overline{Y} + t_{n-1,1-\alpha/2} s / \sqrt{n-1}$$

This equation tells us that the true mean μ falls within the interval

$$\overline{Y} \pm t_{n-1,1-\alpha/2} s / \sqrt{n-1}$$

at a 100(1 − α)% confidence level. Thus, if the level of significance (1 − α) is specified, an appropriate value of $t_{n-1,\,1-\alpha/2}$ can be obtained from Table 1 in appendices. The corresponding confidence interval 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 μ. 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 σ divided by the square root of the sample size *n*. See [Solomon, 1983, p. 229].

Therefore, *N*(0, 1) test is particularly useful for evaluating simulation results. According to the central limit theorem, the statistic

$$(\frac{\overline{Y} - \mu}{s})\sqrt{n-1}$$

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 μ of the random variable falls within the interval

$$\overline{Y} \pm Z_{0.5-\alpha/2}\ s / \sqrt{n-1}$$

at a 100(1 − α)% confidence level. Where $Z_{0.5-\alpha/2}$ is a tabulated valued of the *N*(0, 1) distribution as shown in Table 2 in appendices. For example, given α = 0.05, then

$$Z_{0.5-\alpha/2} = Z_{0.475} = 1.96$$

9.4.4 χ^2 Test

Before a simulation run of the Espresso WfMS, the distribution for creating process instances at a start activity of a process definition, 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/Pooch, 1980, p. 70]

The χ^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 χ^2 statistic can be determined as

$$\chi^2 = (O_1 - E_1)^2/E_1 + (O_2 - E_2)^2/E_2 + \ldots + (O_k - E_k)^2/E_k$$

with $k - 1$ degrees of freedom.

It should be noted that the χ^2 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/Gnugnoli, 1972], [Graybeal /Pooch, 1980, p. 71] and [Solomon, 1983, p. 23].

This statistic is used in conjunction with a χ^2 table, as given in Table 3 in appendices. The rejection probability α given in the top row of the table indicates the probability of incorrectly rejecting the assumed distribution as shown in Figure 9-11, where v is the degrees of freedom.

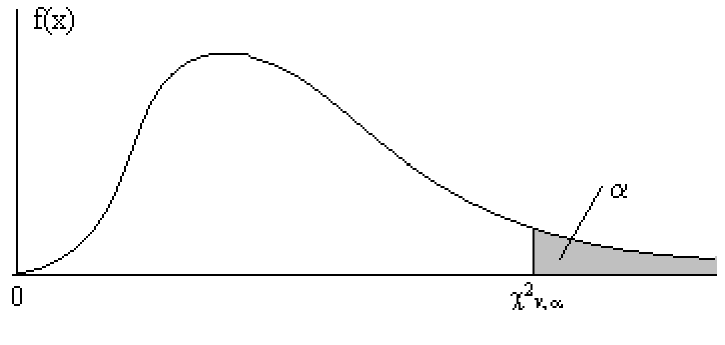


Figure 9-11. The Critical Region of χ^2 Distribution

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 (i.e. that the distribution is inappropriate) increases as the calculated value for χ^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 χ^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 χ^2 value is smaller than the tabulated value for some fairly large rejection probability (e.g. $\alpha = 0.95$ 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 χ^2 value falls within the confidence 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 *F*-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/Gnugnoli, 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 tested statistic *F*-ratio is given by

$$F = s_1^2/s_2^2, \text{ with } n_1 - 1 \text{ and } n_2 - 1 \text{ degrees of freedom, if } s_1^2 > s_2^2$$

or

$$F = s_2^2/s_1^2, \text{ with } n_2 - 1 \text{ and } n_1 - 1 \text{ degrees of freedom, if } s_2^2 > s_1^2$$

For example, if given $F = 1.20$ with 15 and 15 degrees of freedom for $\alpha = 0.05$, from Table 4 in appendices, 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 was observed on some randomly chosen days between 8/17/99 to 9/29/99. One hundred of process intercreation times, denoted respectively by Y_i for $i = 1, 2, \ldots, 100$, were 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
9/23/99:	38, 19, 4, 2, 6, 2, 25, 108, 11, 142
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 μ and variance σ^2 can be estimated respectively from

$$\overline{Y} = (Y_1 + Y_2 + \ldots + Y_{100})/100 = 37.79$$
$$s^2 = (Y_1^2 + Y_2^2 + \ldots + Y_{100}^2)/100 - \overline{Y}^2 = 36.61^2$$

The maximum observed value in the sample is 146. We divide the range of [0, 150) into 10 subintervals of the same time length (= 150/10) and group the 100 values into these subintervals as shown in the following table:

Distribution of Intercreation Time

Category	Subinterval	Observed Frequency	Relative Frequency
1	[0, 15)	32	0.32
2	[15, 30)	28	0.28
3	[30, 45)	10	0.10
4	[45, 60)	10	0.10
5	[60, 75)	5	0.05
6	[75, 90)	4	0.04
7	[90, 105)	2	0.02
8	[105, 120)	2	0.02
9	[120, 135)	4	0.04
10	[135, 150)	3	0.03
Total		100	1

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. Therefore, 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.

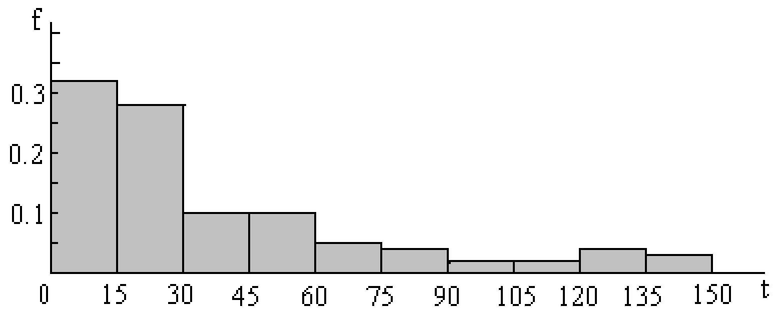
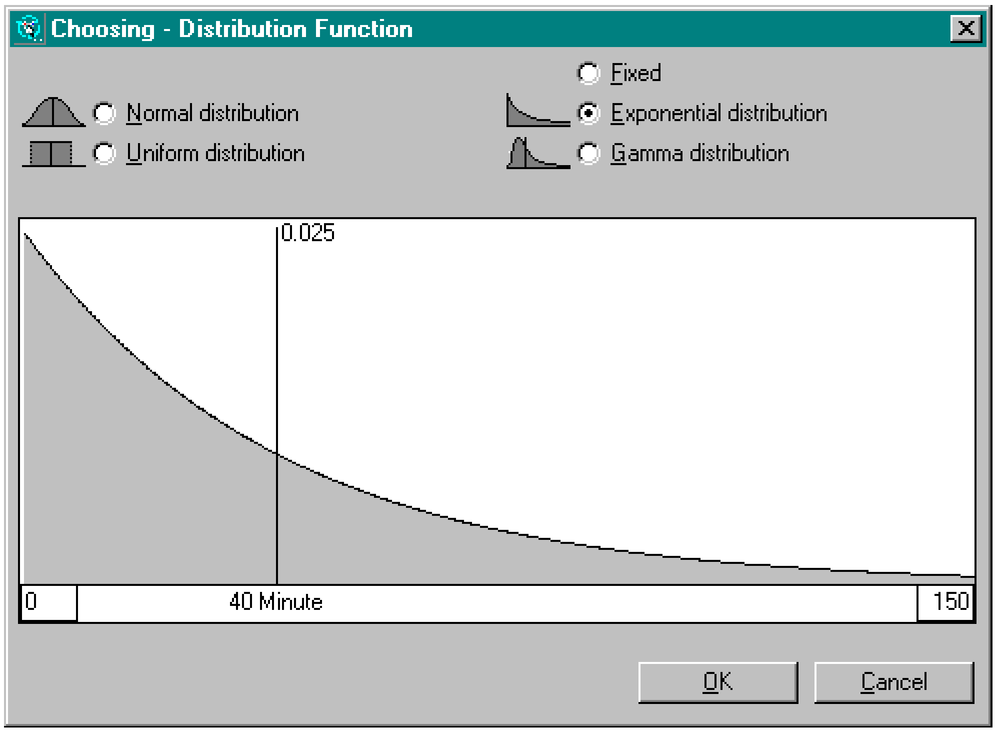


Figure 9-12. Relative Frequency of Observed Intercreation Time



( ) Fixed
( ) Normal distribution
[✓ Exponential distribution]
( ) Uniform distribution
( ) Gamma distribution

0.025
0
40 Minute
150

Figure 9-13. Estimated Distribution of Process Intercreation Time

Because

$$s^2 = 36.61^2 < 40^2 = \sigma^2$$

Thus,

$$F = \sigma^2/s^2 = 40^2/36.61^2 = 1.194, \text{ with } \infty \text{ and } 100 - 1 \text{ degrees of freedom.}$$

From Table 4 in appendices, 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 σ^2 (= 40^2). That is, the hypothesis that the intercreation time has standard deviation 40 is acceptable at a 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

$$\text{Prob}(0 \le X \le x) = \text{Prob}(X \le x) = F(x) = 1 - e^{-\alpha x}$$

Here $\alpha = 1/\mu = 1/40$. So

$$\begin{aligned}\text{Prob}(a \le X < b) &= \text{Prob}(0 \le X < b) - \text{Prob}(0 \le X < a)\\ &= e^{-a/40} - e^{-b/40}\end{aligned}$$

Expected frequency of each category with subinterval $[a, b)$ are compared with the observed frequency in the following table.

Distribution of Intercreation Time vs Expected Frequency

Category i	Subinterval $[a, b)$	Observed Frequency O_i	Expected Frequency $E_i (= n\,(e^{-a/40} - e^{-b/40}))$
1	[0, 15)	32	31.27 (= 100(1.0000 – 0.6873))
2	[15, 30)	28	21.49 (= 100(0.6873 – 0.4724))
3	[30, 45)	10	14.77 (= 100(0.4724 – 0.3247))
4	[45, 60)	10	10.16 (= 100(0.3247 – 0.2231))
5	[60, 75)	5	6.97 (= 100(0.2231 – 0.1534))
6	[75, 90)	4	4.80 (= 100(0.1534 – 0.1054))
7	[90, 105)	2	3.30 (= 100(0.1054 – 0.0724))
8	[105, 120)	2	2.26 (= 100(0.0724 – 0.0498))
9	[120, 135)	4	1.56 (= 100(0.0498 – 0.0342))
10	[135, 150)	3	1.07 (= 100(0.0342 – 0.0235))
Total		100	

Since expected frequencies in categories 6 to 10 are less than 5, we combine them together to category 6 and get the table:

Goodness-of-Fit Test of the Distribution for Intercreation Time

Category i	Subinterval	Observed Frequency O_i	Expected Frequency E_i	$(O_i - E_i)^2/E_i$
1	[0, 15)	32	31.27	0.0170
2	[15, 30)	28	21.49	1.9721
3	[30, 45)	10	14.77	1.5405
4	[45, 60)	10	10.16	0.0025
5	[60, 75)	5	6.97	0.5568
6	[75, ∞)	15	15.34	0.0075
Total		100	100	4.0964

Now we have the χ^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 χ^2 value corresponding to $v = 5$ and $\alpha = 0.05$ is 11.07 (see Table 3 in appendices). Since the calculated value does not exceed this quantity, we accept the hypothesis. Moreover, the tabulated χ^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 a 5% rejection probability.

Now we can hypothesize at the level of significance 95% 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 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 distributions have been used to model “sudden and catastrophic” failures and to characterize service times and interarrival times in queuing systems. The gamma distribution is often used to represent corresponding empirical data. These distribution functions can be utilized directly in a simulation study of the Espresso WfMS.

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—grouped data falling into each subinterval of the sample space. The cumulative distribution of an empirical distribution can be constructed by the data.

If it is hypothesized that a random variable can be derived from some known common distributions, statistical methods can then be used to assess the validity of the hypothesis at 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 χ^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 discussions about hypothesis tests see [Maisel/Gnugnoli, 1972].

10 RANDOM VARIATE GENERATION

10.1 Introduction

*Random number*s 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 the simulation model of a WfMS, some input variables will be decided by unexpected factors and it is not realistic to set them as fixed or deterministic. When simulating the Espresso WfMS, random variates obtained upon random numbers can be used

- to generate activity execution/delay time,
- to generate material occupation time,
- to generate routing time of a work from one activity to another,
- to generate intercreation time of process instances at a start activity of a process definition,
- to determine dynamic role parameter,
- to determine workflow participants among a team, 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 therefore called *pseudorandom number*s.

Ideally, a pseudorandom number generator should possess all of the following desirable characteristics.

- Randomness. First and foremost, the generated sequence of pseudorandom numbers must exhibit the same properties as truly random numbers.
- 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 least be sufficiently large so that the random numbers do not repeat themselves during any single simulation run.

- Reproducibility. For debugging a simulation program or carrying out a parametric study (i.e. varying the operating policy), it may be desirable to generate exactly the same sequence of random numbers during each simulation run. 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 system analyst.
- 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 Numbers

There are many methods to generate random numbers. Here only the most common and simple one will be introduced.

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])

$$n_i \equiv a n_{i-1} (\text{mod } m)$$

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 a known integer constant n_0, called *seed*, then the repeated use of above equation results in

$$\begin{aligned} n_1 &\equiv a n_0 (\text{mod } m) \\ n_2 &\equiv a^2 n_0 (\text{mod } m) \\ &\dots \\ n_i &\equiv a^i n_0 (\text{mod } m) \end{aligned}$$

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 described here.

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 could be 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, if the multiplier satisfies the following two conditions:

   $a \cong 2^{w/2}$ and
   $a \equiv \pm 3 (\bmod 8)$ or $a (\bmod 8) = \pm 3$

3. The seed n_0 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 ($i = 1, 2, \ldots$) obtained in this manner will range from 1 to $(m - 1)$. The desired uniformly distributed random variate u_i ($i = 1, 2, 3, \ldots$) can then be obtained from

$$u_i = n_i / m$$

so that

$$0 < u_i < 1$$

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 there is an example of bad choice of the parameters in accordance with the above rules (see [Fishman, 1973, pp. 176-178]). For more sophisticated methods to generate random numbers see [Fishman, 1973], [Gottfried, 1984], [Maisel /Gnugnoli, 1972] and [Naylor/Balintfy/Burdick/Chu, 1966].

10.2.2 Algorithm for Generating Random Numbers

Hypothesis
A computer has a 16-bit word.

Principle
The power residue method

$$n_i \equiv a n_{i-1} (\text{mod } m) \text{ as well as}$$
$$u_i = n_i / m$$

is to be implemented on the computer with $w = 16$ to generate random numbers. So it can be determined that

$$m = 2^{w-1} = 2^{15} = 32768$$
$$a = 259 \cong 2^{w/2} = 2^8 = 256, \text{ with that } a \equiv \pm 3 (\text{mod } 8)$$

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 the 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: $n_0 \leftarrow n_0(\text{mod } m)$ (let $n_0$ be less than $m$);
Step 3: if $n_0(\text{mod } 2) = 1$ ($n_0$ is odd), go to Step 5;
Step 4: $n_0 \leftarrow n_0 + 1$;
Step 5: $n \leftarrow n_0$;
Step 6: $n \leftarrow an(\text{mod } m)$;
Step 7: $u \leftarrow n/m$;
Step 8: stop (return $u$).
```

Example 10-1. Generate Random Numbers
Running the procedure the first time with seed 139 and then 59 times with seed 0, the following 60 random numbers will be obtained:

```
0.098663330078125  0.553802490234375  0.434844970703125
0.624847412109375  0.835479736328125  0.389251708984375
0.816192626953125  0.393890380859375  0.017608642578125
```

```
0.560638427734375 0.205352783203125 0.186370849609375
0.270050048828125 0.942962646484375 0.227325439453125
0.877288818359375 0.217803955078125 0.411224365234375
0.507110595703125 0.341644287109375 0.485870361328125
0.840423583984375 0.669708251953125 0.454437255859375
0.699249267578125 0.105560302734375 0.340118408203125
0.090667724609375 0.482940673828125 0.081634521484375
0.143341064453125 0.125335693359375 0.461944580078125
0.643646240234375 0.704376220703125 0.433441162109375
0.261260986328125 0.666595458984375 0.648223876953125
0.889984130859375 0.505889892578125 0.025482177734375
0.599884033203125 0.369964599609375 0.820831298828125
0.595306396484375 0.184356689453125 0.748382568359375
0.831085205078125 0.251068115234375 0.026641845703125
0.900238037109375 0.161651611328125 0.867767333984375
0.751739501953125 0.700531005859375 0.437530517578125
0.320404052734375 0.984649658203125 0.024261474609375
```

10.2.3 Generating Random Numbers in Basic Language

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. The utilizations of the random number generator are illustrated in the examples below.

Example 10-2. Generate Random Numbers in Basic
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
0.6035443    0.8158741    0.6778471    0.5325409    0.834121
0.7698958    0.1285262    0.4922713    0.2696272    0.7692857
0.4626319    0.7431279    0.007535756  0.3341243    0.6824226
0.5908359    0.1931716    0.8409376    0.2208206    0.04912817
0.8478358    0.03832173   0.3945374    0.4803704    0.9469314
0.8125682    0.1071984    0.4861861    0.7480877    0.2032888
0.8024682    0.04808176   0.3692856    0.6690407    0.6671755
0.9180714    0.6169828    0.6029513    0.08443779   0.259302
0.06423813   0.09834337   0.3367869    0.5366784    0.3788374
0.96436      0.5592937    0.6574882    0.9014178    0.9915895
0.9426929    0.3620268    0.4735931    0.4676907    0.1865086
0.8642224    0.5160785    0.9430175    0.2021787    0.7422611
0.3568161    0.2462442    0.06618017   0.07921159   0.8090993
0.7499905    0.2231542    0.8536507    0.999698     0.8864462
```

Different runs of the program result in different sequences of 100 pseudorandom numbers.

Example 10-3. Generate Same Sequence of Random Numbers in Basic*
This Basic program repeats sequences of 10 random numbers to a given seed s (> 0):

```
Rnd (−1) * s
For i = 1 To 10
        Let u = Rnd
        Print u
Next i
```

The first "Rnd" function in the program has a negative argument ($-s$), and this makes the statement "Rnd" generate a fixed number corresponding 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 with that $s = 0.4253501$:

```
0.7326744    0.3100755    0.3004674    0.09900606   0.5817471
0.9359531    0.1535475    0.5976273    0.1622995    0.2150481
```

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 the 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 in order 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/Pooch, 1980, p. 86]

Example 10-4. Frequency Test of Random Numbers

The 100 pseudorandom numbers generated in Example 10-2 can be subdivided from the interval (0, 1) into 10 subintervals of the equal width (= 0.1). 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 10 (= 100/10) for each category.

Calculate a χ^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.

Goodness-of-Fit Test of 100 Pseudorandom Numbers

Category n	Subinterval	Number of Observations O_n	Expected Number of Observations E_n
1	(0.0, 0,1)	13	10
2	[0.1, 0.2)	7	10
3	[0.2, 0.3)	7	10
4	[0.3, 0.4)	13	10
5	[0.4, 0.5)	7	10
6	[0.5, 0.6)	7	10
7	[0.6, 0.7)	10	10
8	[0.7, 0.8)	8	10
9	[0.8, 0.9)	12	10
10	[0.9, 1.0)	16	10
Total	(0.0, 1.0)	100	100

The χ^2 statistic is determined as

$$\begin{aligned}\chi^2 &= (O_1 - E_1)^2/E_1 + (O_2 - E_2)^2/E_2 + \ldots + (O_{10} - E_{10})^2/E_{10}\\ &= (13-10)^2/10 + (7-10)^2/10 + (7-10)^2/10 + (13-10)^2/10\\ &\quad + (7-10)^2/10 + (7-10)^2/10 + (10-10)^2/10 + (8-10)^2/10\\ &\quad + (12-10)^2/10 + (16-10)^2/10\\ &= 9.8\end{aligned}$$

Since there are 10 categories, and hence $v = 9$. The tabulated χ^2 value corresponding to $v = 9$ and $\alpha = 0.05$ is 16.92 (see Table 3 in appendices). For the reason that the calculated value does not exceed this quantity, we accept the hypothesis at a 95% confidence level. Moreover, the tabulated χ^2 value corresponding to $v = 9$ and $\alpha = 0.95$ is 3.325. For 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.

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 χ^2 value is less than the tabulated value for a high rejection probability).

10.3.2 Increasing and Decreasing Runs

“The random oscillatory nature of sequences of pseudorandom numbers can be tested by ‘tests of runs’.” [Naylor/Balintfy/Burdick/Chu, 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 $n = 1, 2, \ldots$. 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

$$E_{\mathrm{TOT}} = (2N - 1)/3$$

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 χ^2 statistic based upon the runs of length n.

<u>Example 10-5. Increasing and Decreasing Runs</u>
Let us apply the test of increasing and decreasing runs 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+	0.03868878–
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+
0.6035443–	0.8158741+	0.6778471–	0.5325409–	0.834121+
0.7698958–	0.1285262–	0.4922713+	0.2696272–	0.7692857+
0.4626319–	0.7431279+	0.007535756–	0.3341243+	0.6824226+
0.5908359–	0.1931716–	0.8409376+	0.2208206–	0.04912817–
0.8478358+	0.03832173–	0.3945374+	0.4803704+	0.9469314+
0.8125682–	0.1071984–	0.4861861+	0.7480877+	0.2032888–

0.8024682+	0.04808176–	0.3692856+	0.6690407+	0.6671755–
0.9180714+	0.6169828–	0.6029513–	0.08443779–	0.259302+
0.06423813–	0.09834337+	0.3367869+	0.5366784+	0.3788374–
0.96436+	0.5592937–	0.6574882+	0.9014178+	0.9915895+
0.9426929–	0.3620268–	0.4735931+	0.4676907–	0.1865086–
0.8642224+	0.5160785–	0.9430175+	0.2021787–	0.7422611+
0.3568161–	0.2462442–	0.06618017–	0.07921159+	0.8090993+
0.7499905–	0.2231542–	0.8536507+	0.999698+	0.8864462–

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 $E_{\mathrm{TOT}} = (2N - 1)/3 = 66.3$.

n	O_n	E_n
1	41	41.75
2	18	18.10
3	6	5.15
4	1	1.11
5-99	0	0.23

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

n	O_n	E_n
1	41	41.75
2	18	18.10
3-99	7	6.49

A χ^2 statistic can now be calculated as

$$\chi^2 = (41 - 41.75)^2/41.75 + (18 - 18.10)^2/18.10 + (7 - 6.49)^2/6.49$$
$$= 0.0541$$

Table 3 in appendices indicates a value of 9.210 for $v = 2$ and $\alpha = 0.01$. Since this value exceeds the calculated value, we conclude at a 99% confidence level that the given pseudorandom numbers are sequenced randomly. This conclusion is further supported by the tabulated χ^2 value of 0.00201, corresponding to $v = 2$ 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, there are following tests for various purposes (see [Naylor/Balintfy/Burdick/Chu, 1966, pp. 57-62]).

- *Serial test*: check the degree of randomness between successive numbers in a sequence.
- *The lagged product test*: measure the independence of pseudorandom numbers.
- *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*: concern with the randomness of the digits in a sequence of numbers.
- *The maximum test*: is a more stringent test than the basic frequency test.
- *The poker test*: is 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 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, determine a value in an interval for a random variable following an empirical distribution, and generate the points in time between the specified range for creating initial process instances.
- Normal distribution: generate time.
- Exponential distribution: generate time.
- Gamma distribution: generate time.

- Empirical distribution: determine the interval of a variable, choose workflow participants among a team, decide routing work along one of "Exclusive Choice" outgoing links of an activity, and specify a role parameter.
- *U*[1, *n*] distribution (an integer-valued uniform distribution in the range [1, *n*]): determine number of workflow participants to execute an activity, and the number of values for a multi-value variable.
- 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 or not, and whether the work is divisible by multiple workflow participants of an activity or not.

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 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 *F*(*x*) can be obtained. Thus,

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

where $0 \le F(x) \le 1$. 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.

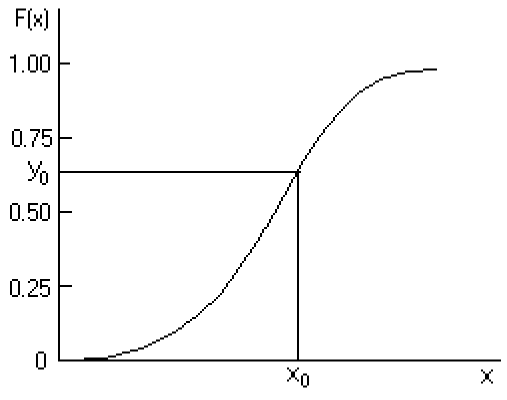


Figure 10-1. The Inverse Transformation Method

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

$$\text{Prob}(Y \leq y_0) = y_0,\ 0 \leq y_0 \leq 1$$

This is the expression for the cumulative distribution of the standard uniform distribution $U(0, 1)$ (see Section 9.2.1). It tells us that Y is uniformly distributed within the interval [0, 1], regardless of the distribution of X. See [Gottfried, 1984, pp. 76-78].

Thus, 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/Pooch, 1980, p. 89]

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

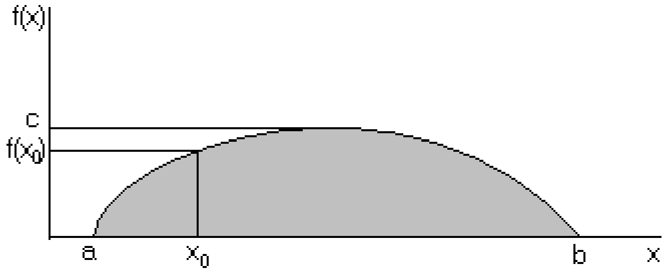


Figure 10-2. The Rejection Method

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 $y_0 = cu_2$;
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

$$x = a + (b - a)\, u$$

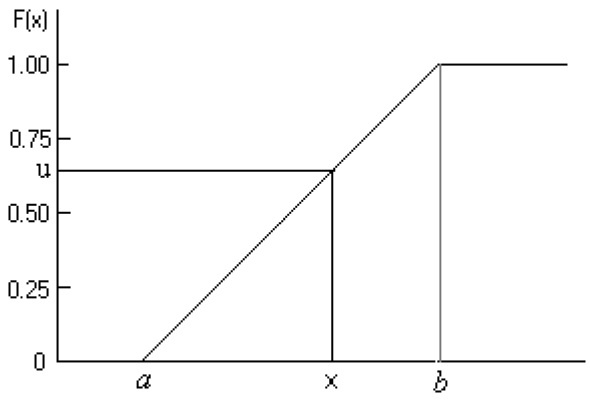


Figure 10-3. $U(0, 1)$ Cumulative Distribution

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 $[a, b]$. That is, X follows $U[a, b]$ distribution. Thus, X can take on the values a, $a + 1$, $a + 2$, …, $b - 1$, b. If u is a random number, then a variate x of X can be obtained by

$$x = a + \mathrm{INT}[(b - a + 1)u]$$

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

$$\mathrm{INT}[(b - a + 1)u]$$

will take on the integer values 0, 1, 2, … , $(b - a)$, and hence X will assume the values a, $a + 1$, $a + 2$, …, 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 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, ..., u_n are a sample of random numbers, then

$$x = \frac{\sigma}{\sqrt{n / 12}} \sum_{i=1}^{n} u_i + (\mu - \frac{n}{2} \frac{\sigma}{\sqrt{n / 12}})$$

is a random observation from an approximately normal distribution with mean μ and standard deviation σ. 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/Lieberman, 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 μ and deviation σ.

Principle
Mean μ and deviation σ are parameters of the procedure. By applying the central limit theorem, a $N(\mu, \sigma)$ variate x can be generated with the above equation, or with

$$x = \mu + \sigma Z, \text{ and}$$

$$Z = (\sum_{i=1}^{n} u_i - \frac{n}{2}) / \sqrt{n / 12}$$

Here u_1, u_2, ..., u_n are a sample of random numbers and n is the size of sample. Let n = 12, we have

$$Z = (u_1 + u_2 + ... + u_{12} - 6)$$

```
Procedure (μ, σ)
Step 1: generate random numbers $u_1, u_2, ..., u_{12}$;
Step 2: $Z \leftarrow (u_1 + u_2 + .... + u_{12} - 6)$;
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 to generate exponentially distributed random variates, the equation

$$F(x) = 1 - e^{-\alpha x}$$

must be solved for x first. Consequently,

$$x = -(1/\alpha) \ln[1 - F(x)]$$

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,

$$x = -(1/\alpha) \ln u$$

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, the exponential distribution can be used to generate random time with given mean time. Suppose that random variable X is exponentially distributed, a variate x of X can be generated with

$$x = -(\mu) \ln u$$

Here μ 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 (i.e. $0 < x_0 \le x$). The equation for generating the variates must be modified to (see [Gottfried, 1984, p. 86])

$$x = x_0 - (1/\alpha) \ln u$$

Also, the relationship between α and μ 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 β exponential random variates. Thus, we obtain

$$x = -\ (1/\alpha)\ln\left(\bigcap_{i=1}^{\beta} u_i\right)$$

where x is the desired variate of a random variable following the gamma distribution with parameters α and integer-valued β, 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

The 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 μ and deviation σ. The gamma distribution has the parameter β as integer.

Principle
Mean μ and deviation σ are parameters of the procedure. A gamma variate x will be returned by this procedure. Because

$$\mu = \beta/\alpha \text{ and } \sigma^2 = \beta/\alpha^2 = \mu/\alpha$$

and β is an integer, so it is assumed that

$$\beta = \mathrm{INT}(\mu^2/\sigma^2)$$

Now

$$\alpha = \beta/\mu$$

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 the calculation.

```
Procedure (μ, σ)
Step 1: let $\beta \leftarrow \mathrm{INT}(\mu^2/\sigma^2)$;
Step 2: if $\beta < 1$, let $\beta \leftarrow 1$;
Step 3: let $\alpha \leftarrow \beta/\mu$;
Step 4: let $p \leftarrow 0$ and $i \leftarrow 1$;
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 $i \le \beta$, 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 μ and standard deviation σ, β should not always be handled as integer-valued.

Hypothesis

It is known that a random variable X is gamma distributed with mean μ and deviation σ. The gamma distribution has the parameter β, which can be a non-integer.

Principle

Suppose that random variable X is gamma distributed with β not always being an integer. The density function is then

$$f(x) = \frac{\alpha^\beta x^{(\beta-1)}}{\Gamma(\beta)} e^{-\alpha x}, \quad for\ x \ge 0$$

A variate x of X can be generated as described in the procedure (see [Fishman, 1973, pp. 208-210]).

Furthermore, because

$$\mu = \beta/\alpha \text{ and } \sigma^2 = \beta/\alpha^2 = \mu/\alpha$$

so

$$\beta = \mu^2/\sigma^2 \text{ and } \alpha = \beta/\mu$$

Mean μ and deviation σ are parameters of the procedure. Temporary variables k, Y, Z, γ, j, A, B are used in the calculation.

```
Procedure (μ, σ)
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:  $k \leftarrow \mathrm{INT}(\beta)$;
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 $A + B \le 1$, 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 through Step 14 is a rejection method to generate a variate following Beta distribution with parameter γ and $1 - \gamma$.

10.4.6 Empirical Distributed Random Variates

Variates of a random variable X following an empirical distribution, as shown in Figure 9-5 of Section 9.2.5, can be obtained through the inverse transformation method, while the cumulative distribution Y_j as shown in Figure 9-6 can easily be constructed from f_j, the given probability that the value of X following the empirical distribution will fall into the jth subinterval.

Suppose that XL_j and XU_j, $j = 1, 2, \ldots, m$, are the lower and upper interval bounds of the jth subinterval respectively. It is known that

$$\text{Prob}(X \le XU_j) = Y_j$$

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

$$XL_j \le X \le XU_j, \text{ if } Y_{j-1} < u \le Y_j$$

That is, the value of X falls into the jth subinterval $[XL_j, XU_j]$ if $Y_{j-1} < u \le Y_j$.

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 jth subinterval. $f_1 + f_2 + \ldots + f_m = 1$. XL_j and XU_j are the lower and upper bounds of the jth 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 $u >$ 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 χ^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 χ^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 χ^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 μ and standard deviation σ. 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 χ^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 the $U(0, 1)$ distribution will be required for a simulation run of a WfMS. The random numbers generated by a computer program are pseudorandom numbers. If a general-purpose programming language, which is used to build up a simulation model, does not offer a random number generator, some determined calculations, such as the power residue method, could be used to generate the random number sequence. An ideal random 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 various 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 χ^2 test.

From pseudorandom numbers, random variates of other distributions can be generated. 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 (simulator) integrated in PM, diverse process definitions implemented in different Espresso application databases can be simulated simultaneously. There are many 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. During simulation, system states and process protocols can be visually displayed.

11.1 Process Settings

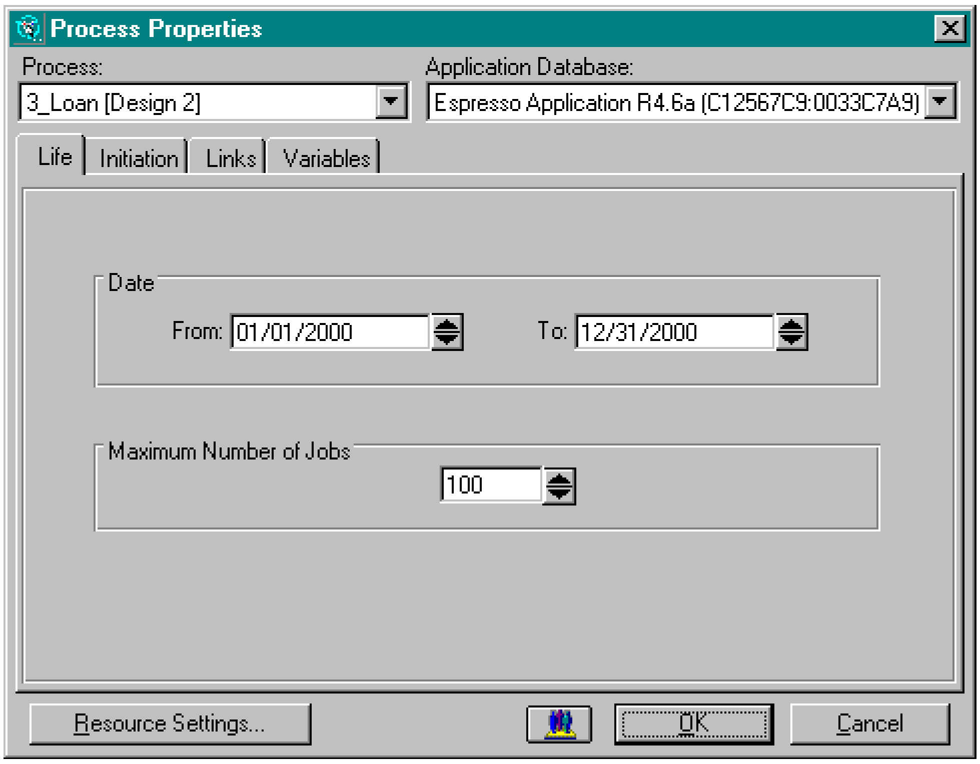
Various input variables associated with a process definition are specified via the process settings. They are utilized in the simulation study of the Espresso WfMS,

- to allow a process definition to be implemented only in a certain period of time,
- to create process instances according to the given distribution specified for each start activity of a process definition,
- to route a work along a “Multiple Choice” link with a given probability,
- to route a work along one of the “Exclusive Choice” outgoing links of an activity following a given empirical distribution, and
- to determine the value of a variable governed by the given distribution.

The process settings are used by the simulator and saved in the Espresso Simulation Database.

11.1.1 Process Life Period Settings

A process definition has a network of activities and is used for the 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. Therefore, it is sometimes necessary to specify for a process definition the maximum number of process instances (jobs), and/or the life beginning date as well as the life ending date during which the process instances can be created. The dialogue box in Figure 11-1 is offered by PM for the life specification relevant to an Espresso application database.


Process Properties

Process: 3_Loan [Design 2]

Application Database: Espresso Application R4.6a (C12567C9:0033C7A9)

Date

From: 01/01/2000 To: 12/31/2000

Maximum Number of Jobs: 100

Figure 11-1. Process Life Period Settings

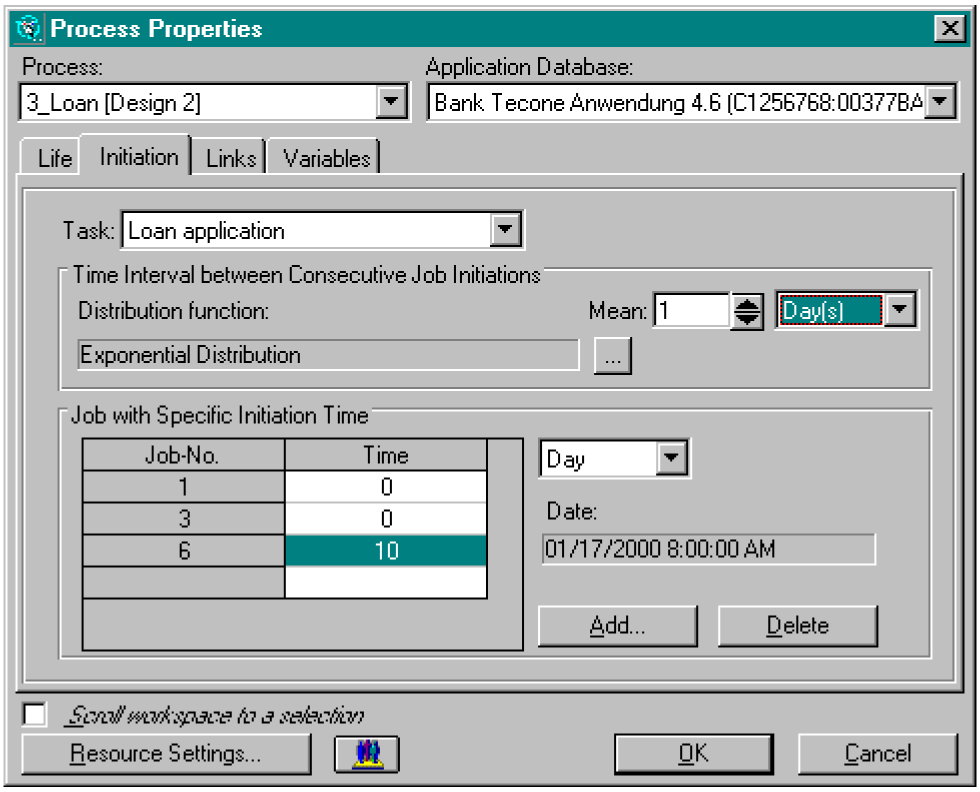
For the simulation study, the life beginning date determines when the process instances of a process definition can start to be simulated. The maximum number of the process instances and life ending date determine when to stop creation of the process instances. Creation of process instances in the simulated Espresso application database stops when one of the stipulated termination conditions is met.

Life ending date works together with the simulation beginning date. If only the process instances of one process definition in a single Espresso application database are simulated, 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 process definitions in all simulated Espresso application databases.

11.1.2 Process Creation Settings

In the real world 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) specified to the start activities of the process definition. For the simulation study, the rule used by the simulator for creating process instances should be given.

Before a simulation run, the system analyst should specify for each start activity (task) of a process definition the distribution function as well as the parameters (i.e. mean time and standard deviation) in order to generate process intercreation time of the start activity in a simulated Espresso application database (see Figure 11-2). According to the given function of exponential,



Job-No.	Time
1	0
3	0
6	10

Figure 11-2. Process Creation Settings

gamma, uniform or normal distribution, the simulator generates random variates of the process intercreation time for determining when the next process instances in the application database will be created at the start activity.

The process intercreation 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), in order to validate the simulator, evaluate the simulation results, compare alternative operating policies, etc.

If a business process in accordance with a process definition is newly implemented, it is possible that the instances in the initial period will not be created as regularly as that specified with the distribution. Therefore, PM allows the system analyst to give specific points in time for creating initial process instances (jobs) of the 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 (e.g. process 3 and process 6 in Figure 11-2—one will be created at the beginning of a simulation run and another on the tenth day), the creation times for the process instances between the two neighboring given process instances but not numbered in succession (i.e. process 4 and process 5), will be generated according to the uniform distribution in the range between the given points in time for creating the two neighboring given process instances (i.e. according to

U(0, 10) distribution—process 4 may be determined to be created on the 3rd day and process 5 on the 7th day, and 3 and 7 are clearly within range [0, 10]).

A *process creation event* is an arrival event that will be further discussed in Section 13.4. When a process creation event occurs, a new process instance will be created in a simulated WfMS. A process creation event is scheduled according to the process creation settings for each start activity of a process definition in a simulated application database.

Assumption 11-1. Simulating Process Creations
For each start activity of a process definition whose process instances will be simulated in an application database, the 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; 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 in 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 points in time for creating process i (≥ 0) and process k ($> i + 1$) are given as a and b respectively but that for creating processes $i + 1$, $i + 2$, …, and $k - 1$ are not given, the points in time for creating the not given processes are assumed following the $U(a, b)$ distribution. Here it is assumed that $a = 0$ for $i = 0$ (when the point in time for creating the first process instance is not given).

From Assumption 11-1, for each start activity of an analyzed process definition, a distribution function will be utilized by the simulator 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 the specific point in time is created.

The process creation settings are used to initialize the eventlist of the Espresso simulation model at the beginning of a simulation run. Thus, a simulation run can go further with a non-empty eventlist.

Example 11-1. Generate Process Creation Time
Start activity "Register order" of the process definition in the Figure 1-3 of Section 1.3 is specified with the process creation settings as shown in Figure 11-2.

The process intercreation function is an exponential distribution with mean one day. The specific points in time for creating processes 1, 3, and 6 are given as: process 1 and process 3 are created at the beginning time of implementing the process definition; process 10 is created on the 10th day.

We simulate the process instances of the process definition two times and gather the generated points in time for creating the first ten process instances respectively for each run. The data are presented in the following table (1 day = 8 hours = 480 minutes).

Process Instance No.	Creation Time (minute) (Run 1)	Creation Time (minute) (Run 2)
1	(given) 0.	(given) 0.
2	0.	0.
3	(given) 0.	(given) 0.
4	3523.	633.
5	4579.	2942.
6	(given) 4800.	(given) 4800.
7	4939.	6163.
8	5220.	6589.
9	5877.	7014.
10	5896.	7426.

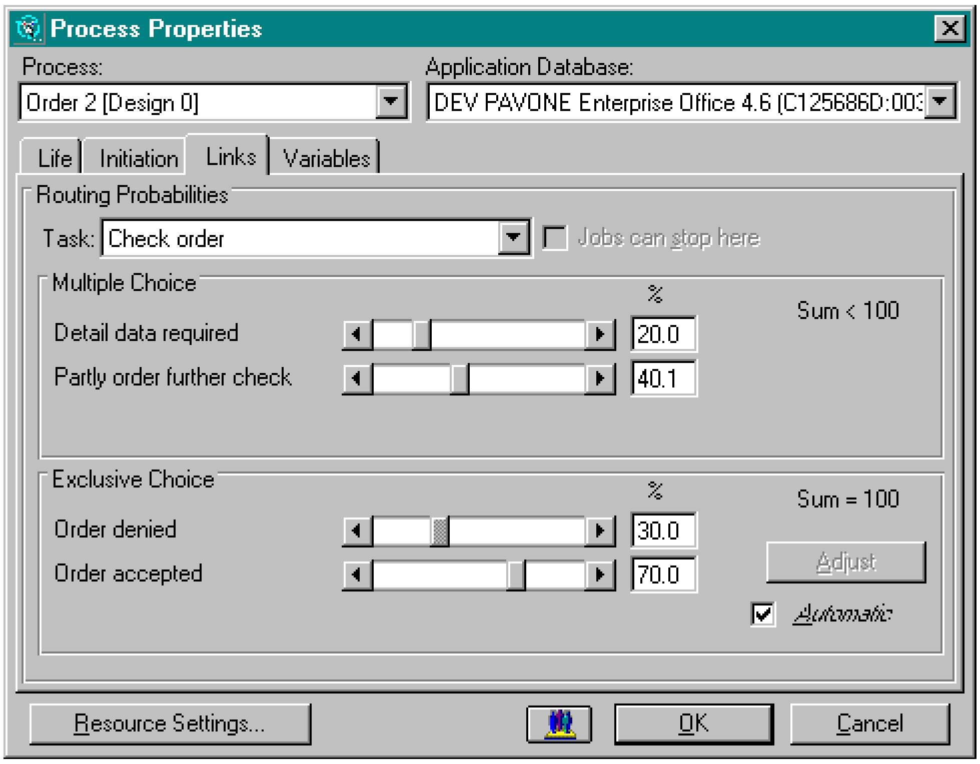
In each run, the point in time for creating process 2 is determined by generating a $U(0, 0)$ variate, and those for creating process 4 and process 5 are determined by generating $U(0, 4800)$ variates. From process 7 on—after creating the last given process instance with the specific point in time (i.e. process 6), the intercreation time, used for generating points in time for creating process instances one after another, 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 points in time in the initial process creation series are relative to it; otherwise they are corresponding to the value of the clock, and are 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 the 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 real world Espresso WfMS, the person who completes an activity decides whether to route the work at the activity along an "Multiple Choice" link or not, and/or along which one of the "Exclusive Choice" outgoing links of the activity to route the work further. However the simulator makes these routing decisions upon the probability settings to the "Multiple Choice" and "Exclusive Choice" outgoing links of an activity as shown in Figure 11-3.


Process Properties

Process: Order 2 [Design 0]

Application Database: DEV PAVONE Enterprise Office 4.6 (C125686D:003

Routing Probabilities

Task: Check order

☐ Jobs can stop here

Multiple Choice — Sum < 100

	%
Detail data required	20.0
Partly order further check	40.1

Exclusive Choice — Sum = 100

	%
Order denied	30.0
Order accepted	70.0

☑ Automatic

Figure 11-3. Process Routing Choice Settings

According to the definitions of routing options in Chapter 3, along only one of all the "Exclusive Choice" outgoing links of an activity a work can be routed further. Therefore, the specification for the "Exclusive Choice" outgoing links of an activity is an empirical distribution for choosing one among them. That is, the sum of routing probabilities for all the "Exclusive Choice" outgoing links of an activity must be 1, or 100%.

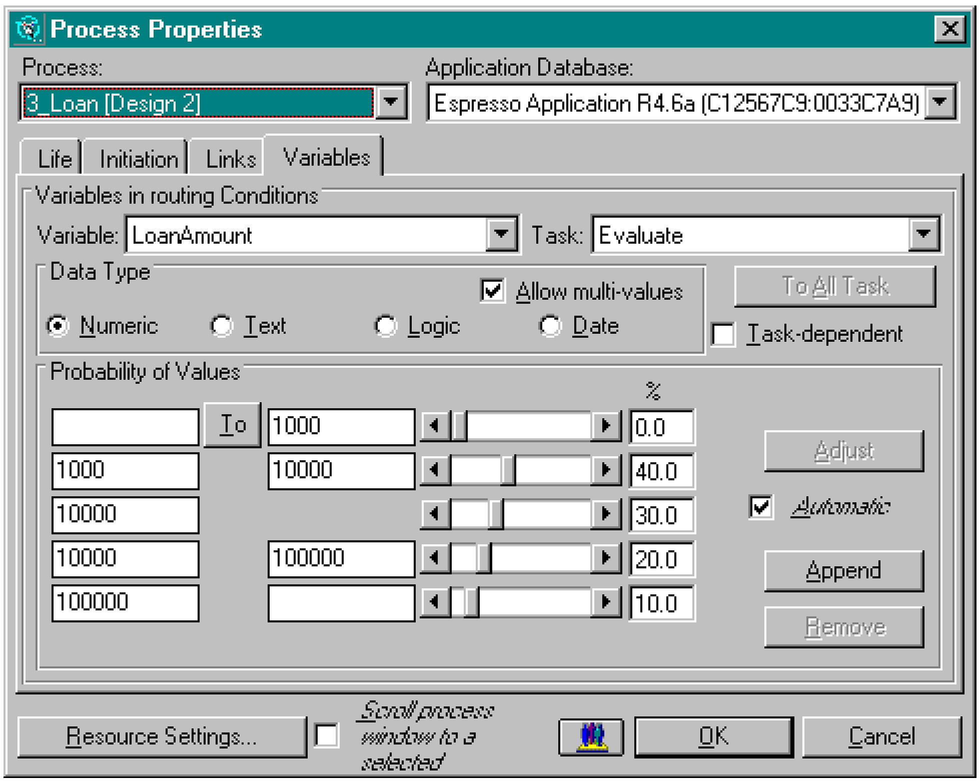
Since a person can choose several "Multiple Choice" outgoing links after completing an activity, whether to route a work along an "Multiple Choice" outgoing link of the activity or not can be assumed to follow a 0-1 distribution in the simulation model. Therefore, routing probability of one "Multiple Choice" outgoing link of an activity dose not influence on those of other "Multiple Choice" outgoing links of the same activity.

Assumption 11-2. Simulating the Routing of a "Multiple Choice" Link
Whether to route a work along a "Multiple Choice" link or not is determined by generating a variate governed by the 0-1 distribution with a given probability. The determination is independent on other "Multiple Choice" outgoing links of the same activity.

11.1.4 Variable Settings

The "Condition" link within a process definition (see Figure 3-1 in Section 3.1) is used by the Espresso workflow engine integrated in a Notes application database to make the routing decision according to the current values of some variables, which stand for Notes fields in the document representing a process instance and can be one of the data types of number, text, logic and date. At run-time, the value of a variable in the application database will be determined according to the circumstances of a particular business process.

For a simulation study, the distribution for determining values of a variable in routing conditions should be specified in the way as shown in Figure 11-4.



Process: 3_Loan [Design 2]
Application Database: Espresso Application R4.6a (C12567C9:0033C7A9)

Variables in routing Conditions
Variable: LoanAmount Task: Evaluate

Data Type
[✓ Allow multi-values]
[✓ Numeric] Text Logic Date
Task-dependent

Probability of Values

			%
	To	1000	0.0
1000		10000	40.0
10000			30.0
10000		100000	20.0
100000			10.0

[✓ Automatic]

Scroll process window to a selected

Figure 11-4. Process Variable Settings

After the data type of a variable has been settled, 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 must 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 data type of a variable is numeric.

If the value distribution of a variable is specified as activity-dependent, the variate of the variable defined in an "Condition" outgoing link of an activity (task) will be generated again according to the corresponding distribution as the activity is completed; otherwise the variate of a variable is generated only once for a process instance, no matter at which activity. So the value distribution of an activity-independent variable is the same at any activity within the process definition.

Assumption 11-3. Simulating Value of 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.

In a Notes database it is possible to allow multi-values separated with ":" to be assigned to a Notes field. In order to simulate this case, the system analyst should specify in the dialogue box shown in Figure 11-4 that the variable is allowed to have multi-values.

Assumption 11-4. Simulating Values of a Multi-Value Variable
If a variable is allowed to be assigned with multi-values in the Espresso WfMS, the values are determined by the simulator in the following steps:

1° n, the number of values, is generated according to $U[1, m]$ distribution (here m is the number of subintervals of the empirical distribution followed by the variable);
2° let n be the same as M, if the generated n is larger than M (here M is the number of subintervals with non-zero probability);
3° according to the specified empirical distribution, n values will be generated in n different subintervals.

For example, from the settings in Figure 11-4, assignment of multi-values is allowed for variable "LoanAmount". Five different subintervals are specified with relevant probabilities. One variate generated according to the distribution can be "82900:1693:592777886" —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 variable settings in Figure 11-4, should not be specified in the value distribution. But according to Assumption 11-4, this increases the number of subintervals of the empirical distribution of a variable and consequently increases the number of multiple values in a value determination of the variable.

Before a simulation run, data types and the value distributions of all variables defined in different routing condition formulas should be specified as they appear in the real world. The simulator can determine whether to route the work along the "Condition" outgoing links of an activity or not, if values of the variables in all the condition formulas can be or have been generated. The default data type of a variable is text.

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 a simulation run that neither a link will be chosen nor a routing condition is met for a work at the activity to flow further. For the simulation study, the system analyst can decide via the dialogue box shown in Figure 11-3 whether a work can stop at the activity.

Suppose that a work cannot stop at a routing decision activity. When the activity is completed, the simulator 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 "Condition" outgoing 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 "Multiple Choice" outgoing link of the activity.

Assumption 11-5. Simulating Stop of a Process Work
At an activity that has only "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 simulator can prompt with a message about a stopped work, and meanwhile let the relevant activity on the process map be square-marked.

If a work is not allowed to stop at an activity that has only "Condition" outgoing links and no variable in the routing formulas of these links is activity-dependent, it can happen that the work can never run further. Therefore, some work items completed there are locked and the associated process instance cannot be terminated. The system analyst should 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 simulator repeats a maximum of 99 times for the determination of routing of a work from such an activity.

11.2 Graphic Process States

During a simulation run, 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

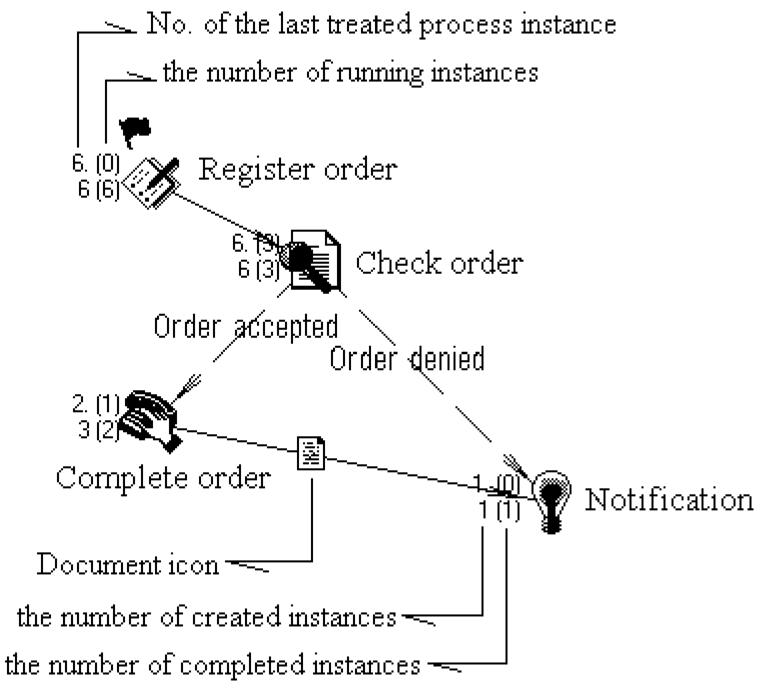

No. of the last treated process instance
the number of running instances
6. (0)
6 (6)
Register order
6. (3)
6 (3)
Check order
Order accepted
Order denied
2. (1)
3 (2)
Complete order
1. (0)
1 (1)
Notification
Document icon
the number of created instances
the number of completed instances

Figure 11-5. Graphical Process States

document icon with the number of the associated process instance on 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 of 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; 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 associated 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”.

Summarily, 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 data of the analyzed process definition.

11.3 Data Structure of Simulated Work Items

If parallel activities are designed in a process definition, it is possible that during a simulation run, 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 must be maintained. A work protocol keeps the start activity, where the associated process instance was 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 the work items are joined together.

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 work records 2, 4, and 5 as shown in Figure 11-6, are connected with one another to a loop (a closed chain) by a pointer of each work record. A parallel work loop helps to locate all parallel work items associated with the same process instance from any given work record in the loop.

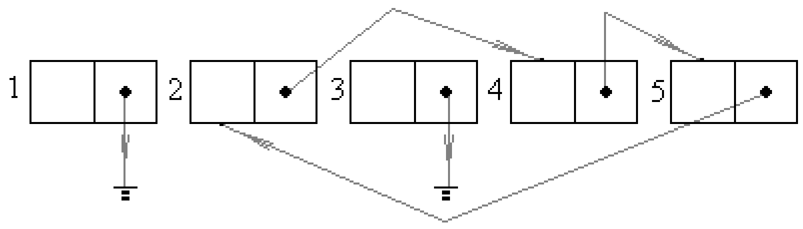


Figure 11-6. Parallel Work Record Loop

Example 11-2. Parallel Work Records
Suppose that 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 work records 2, 4, 5 are associated with one process instance and are connected into a loop 2→4→5→(2). That is, the work in work record 1 is associated with a process instance, the work in work record 3 is associated with another process instance, and the parallel work items in work records 2, 4, and 5 are associated with a common process instance.

Now suppose that the work in work record 4 will be split into two work items, and the new one is kept in work record 6. That is, the new work in work record 6 is parallel with that in work record 4. The pointer from work record 4 connecting to work record 5 is removed and one from work record 4 connecting to work record 6 and another from work record 6 connecting to work record 5 is added, as shown in Figure 11-7.

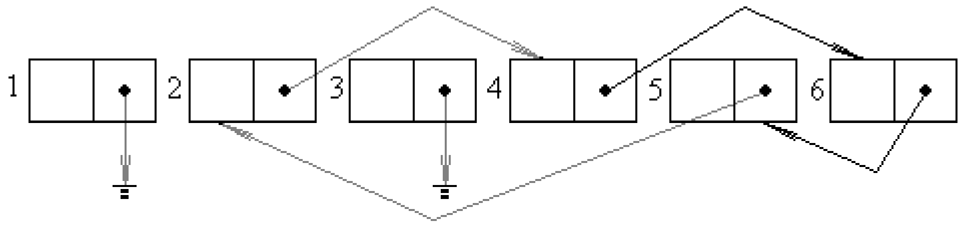


Figure 11-7. Parallel Work Record Loop—Add

At the present, the four parallel work items in loop 2→4→6→5→(2) are associated with the same process instance.

Now suppose that from the work records in Figure 11-7, the parallel work in work record 2 will be removed. Before it is done, the pointer connecting work record 5 to work record 2 will be altered to connect to work record 4, which is originally connected from work record 2. The existing work records will be connected as that shown in Figure 11-8. At

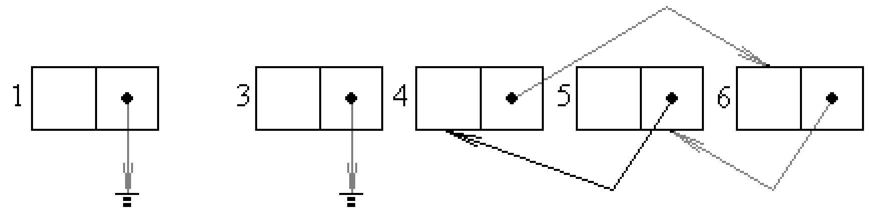


Figure 11-8. Parallel Work Record Loop—Remove

this moment, there are five work items in work records 1, 3, 4, 5, and 6 respectively, associated with three simulated process instances.

The data structure for keeping parallel work items are used by the algorithms for Detecting Deadlocks and Routing 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 a parallel chain as shown in Figure 11-9. The

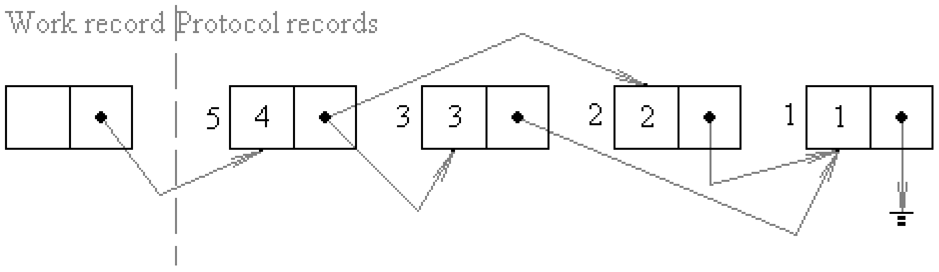


Figure 11-9. Work Protocol Records

work record has a pointer connecting to the protocol record of the activity at which the work is currently located.

A protocol record contains some of workflow control data as well as application data about an activity. The most important data of a protocol record used by the simulator 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 simulator to spot the split activities according to the work protocols of two parallel work items, and to determine critical activities on a work protocol.

Example 11-3. Work Protocol Records
For the process definition in Figure 5-5 of Section 5.2.3, suppose that a work is now at activity 4. The work 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. Protocol record 5, the last protocol record of the work 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 protocol records 2 and 3 have a predecessor set of {1} and protocol record 1 keeps the data of activity 1. Therefore, activity 2 and activity 3 was executed after activity 1 was completed, and hence activity 1 is a split activity for the work. Protocol record 1 has an empty predecessor set and activity 1 is, thus, the start activity of the work kept in the work record.

Based upon the work protocol, a report about the activity execution thread (sequential as well as parallel) of a process instance, called *process protocol* in PM, can be generated from the start activity till the 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.

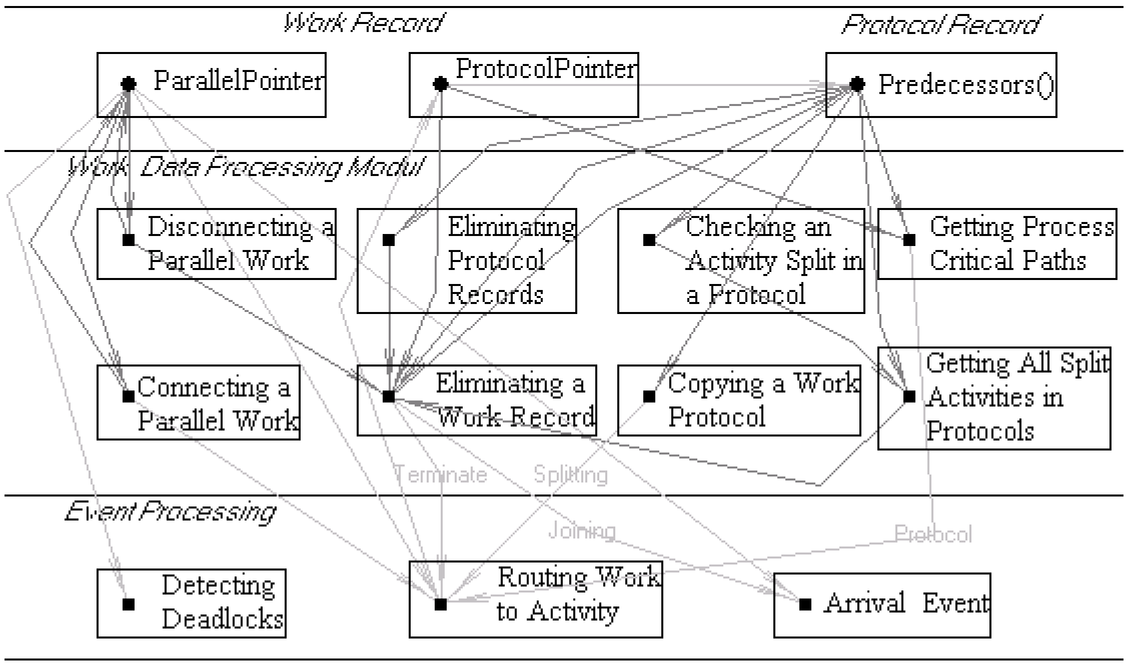

Work Record
Protocol Record
ParallelPointer
ProtocolPointer
Predecessors()
Work Data Processing Modul
Disconnecting a Parallel Work
Eliminating Protocol Records
Checking an Activity Split in a Protocol
Getting Process Critical Paths
Connecting a Parallel Work
Eliminating a Work Record
Copying a Work Protocol
Getting All Split Activities in Protocols
Terminate
Splitting
Event Processing
Joining
Protocol
Detecting Deadlocks
Routing Work to Activity
Arrival Event

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 and before calling algorithm for Routing Work to Activity.

Hypothesis
ParallelPointer(*j*) represents the pointer of work record *j* for connecting parallel work items into a loop.

Principle
Here work records *k* and *i* are parameters of the procedure. The work in work record *k* is newly created and it is parallel with that in work record *i*.

Because the work in work record *k* is parallel with that in work record *i*, work record *k* will be connected to the parallel work record loop with work record *i*. (See Figure 11-6 and Figure 11-7, where the work in work record 6 is newly created and is parallel with that in work record 4 and so it is added to the parallel loop.)

Procedure (*k*, *i*)
```
Step 1: if ParallelPointer($i$) = 0 (the work in work record $i$ is originally not
            parallel), go to Step 3;
Step 2: ParallelPointer($k$) ← ParallelPointer($i$); go to Step 4;
Step 3: ParallelPointer($k$) ← $i$;
Step 4: ParallelPointer($i$) ← $k$; 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
ParallelPointer(*j*) represents the pointer of work record *j* for connecting a parallel work loop.

Principle
Here work 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 work record 2 is removed.)

Temporary variable RecordNo is used for keeping the current treated work record.

Procedure (*i*)

```
Step 1: if ParallelPointer(i) = 0 (the work in work record i is not parallel), stop;
Step 2: RecordNo ← ParallelPointer(i);
Step 3: if ParallelPointer(RecordNo) = i, go to Step 5;
Step 4: RecordNo ← ParallelPointer(RecordNo); go to Step 3;
Step 5: (work record RecordNo is pointing to work record i) if RecordNo =
        ParallelPointer(i) (there is only one remained parallel work to the work
        in work record i), let ParallelPointer(RecordNo) ← 0 and stop;
Step 6: ParallelPointer(RecordNo) ← 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 and before calling algorithm for Routing Work to Activity. For each new created work, a work protocol from the start activity till the current activity must be generated according to the protocol of the original work.

Hypothesis

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 originally called by another procedure when creating a parallel work, which will be routed to an activity kept in protocol record *k*, from the activity kept in protocol record *i*. The work protocol up to the last protocol record *i* (protocol record *i* as well as those of all previously completed activities) must 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 protocol records and must be clear to empty 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:  (protocol 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 protocol 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 record $i$ as well as those of
         previous activities 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
Predecessors(j) represents the predecessor set of protocol record j.

Principle
This procedure is a self-called procedure. Here protocol record i is a parameter of the procedure. The protocol part till protocol record i (i.e. protocol record i as well as all protocol records of the previous activities) will be released.

Global set HasReleased, which keeps just released protocol records, will 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 $i \in$ HasReleased, stop;
Step 2: RestRecords $\leftarrow$ Predecessors($i$);
Step 3: if RestRecords = $\phi$, go to Step 8;
Step 4: remove an element, say protocol record $p$, from RestRecords;
Step 5: if $p \in$ HasReleased, go to Step 3;
Step 6: call the procedure self with parameter $p$;
Step 7: go to Step 3;
Step 8: (all protocol records keeping the previous activities of the activity kept
        in protocol record $i$ have bee released) release protocol record $i$;
Step 9: HasReleased $\leftarrow$ 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
Predecessors(*j*) represents the predecessor set of protocol record *j*.

Principle
Protocol records *i* and *k* are parameters of the procedure. It is known that the activity in protocol record *i* is parallel with an activity in the work protocol part till protocol record *k*.

This self-called procedure checks according to the arrival time whether the activity in protocol record *i* is a split activity in the work protocol part till protocol record *k*. If so, the protocol record of the split activity in the work protocol part will be returned; otherwise NULL 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 in protocol record $i$ and protocol record $k$ 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: RestRecords $\leftarrow$ Predecessors($k$);
Step 5: if RestRecords = $\phi$, stop (return NULL);
Step 6: remove an element, say protocol record $p$, from set RestRecords;
Step 7: $r \leftarrow$ call the procedure self with parameters $i$ and $p$; if $r \neq$ NULL, 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
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. It is known that the work with the protocol part till protocol record *i* is parallel to the work with the protocol part till protocol record *k*.

This self-called procedure puts a set of data about split activities of the two parallel work items, say *i#p#r*, to set *U*, if the activity in protocol record *p*, $p \in$ Predecessors(*i*), is also in protocol record *r* that belongs to the work protocol part till 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. Temporary set RestRecords is used to keep not treated protocol records belonging to the predecessor set of protocol record *i*.

```
Procedure (i, k, U)
Step 1: RestRecords ← Predecessors(i);
Step 2: if RestRecords = $\phi$, stop (set U has the returned value);
Step 3: remove an element, say protocol record p, from set RestRecords;
Step 4: r ← Checking an Activity Split in a Protocol (p, k);
Step 5: if $r \neq$ NULL (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 at an end activity of a process definition, or joined at a join activity.

Hypothesis

Predecessors(*j*) represents the predecessor set of protocol record *j*. ProtocolPointer(*d*) stands for the pointer to the last protocol record that keeps current activity of the work kept in work 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 work records *i* and *k* are parallel (associated with the same process instance) and the work in work record *i* is just coming to the join activity where the work in work record *k* is waiting for joining (both work items are at the same activity now). Therefore, the protocol of the work in work record *i* must be joined to that in work record *k*.

For example, for the process definition in Figure 11-11, the work in work

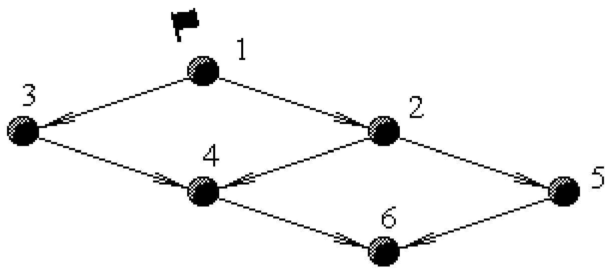


Figure 11-11. Process Definition—Eliminating a Work Record

record *k* came from activity 5 and is waiting at activity 6. Now the parallel work in work 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

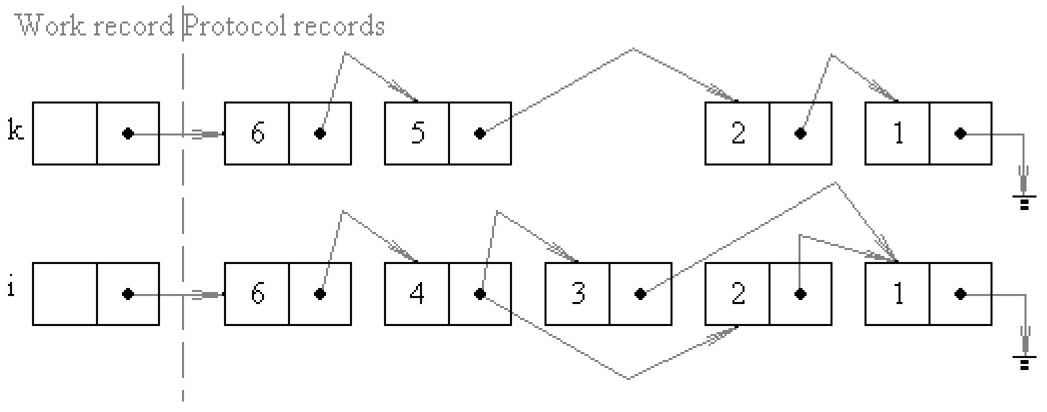


Figure 11-12. Eliminate a Parallel Work Record—before

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

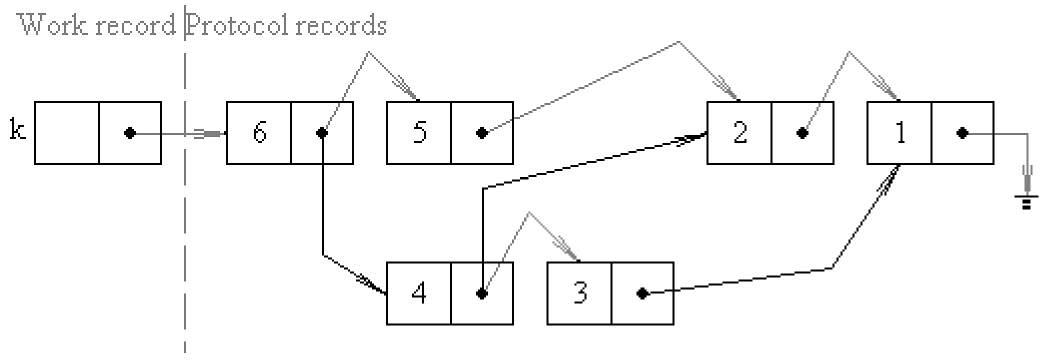


Figure 11-13. Eliminate a Parallel Work Record—after

i and three protocol records connected in the protocol of the work in work record *i* (one keeps the current activity 6, and two others keep split activities 1 and 2) are not used in the new work protocol anymore, and hence will be released.

Temporary set SplitActivityRecs is used in the procedure.

```
Procedure ($i$ , $k$)
Step 1:  $r \leftarrow$ ProtocolPointer($i$) ($r$ is the last protocol record of the work in work record $i$);
Step 2:  if $k \neq$ NULL, go to Step 4 (to merge the protocol of the work in work record $i$ into that in work record $k$);
Step 3:  (the work in work record $i$ is not parallel and so the associated process instance is terminated) HasReleased $\leftarrow \phi$; call Eliminating Protocol Records ($r$); go to Step 14;
Step 4:  $q \leftarrow$ ProtocolPointer($k$) ($q$ is the last protocol record of the work in work record $k$);
Step 5:  if Predecessors($q$) = $\phi$ (activity in protocol record $q$ has no predecessor, so there is no protocol to merge), go to Step 14;
Step 6:  SplitActivityRecs $\leftarrow \phi$; call Getting All Split Activities in Protocols with parameters $r$, $q$ and SplitActivityRecs (set SplitActivityRecs contains message of all split activities where the work has been split);
Step 7:  HasReleased $\leftarrow \phi$ (clear the set that will be used in Eliminating Protocol Records latter in this procedure for eliminating the protocol part of the work in work record $i$, till the protocol record keeping a split activity);
Step 8:  if SplitActivityRecs = $\phi$, 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 $s$ are the same split activity. Protocol records $n$ and $p$ are in the protocol of the work kept in work record $i$, and protocol record $s$ is in the protocol of the work kept in work record $k$);
Step 10: Predecessors($n$) $\leftarrow$ Predecessors($n$) $- \{p\} \cup \{s\}$ (switch a previous protocol record of protocol record $n$ from protocol record $p$ to protocol record $s$);
Step 11: call Eliminating Protocol Records ($p$) (release the protocol part till protocol records $p$);
Step 12: go to Step 8;
Step 13: (to eliminate the last protocol record of the work in work record $i$)
         1° Predecessors($q$) $\leftarrow$ Predecessors($q$) $\cup$ Predecessors($r$) (the predecessor set of the work in work record $i$ is merged to the predecessor set of the work in work record $k$);
         2° release protocol record $r$;
Step 14: call Disconnecting a Parallel Work ($i$);
Step 15: release work record $i$.
```

11.3.3.8 Algorithm for Getting Process Critical Paths

This procedure is used when displaying a process protocol, an activity 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, till the current activity will be determined.

Hypothesis
ProtocolPointer(*i*) represents the pointer connecting to the last protocol record which keeps current activity of the work in work record *i*. Predecessors(*j*) and ArrivalTime(*j*) stand for the predecessor set and work arrival time of the activity kept in protocol record *j*.

Principle
Here work record *i* is a parameter of the procedure. In this procedure, OnCritical(*j*) will be set to TRUE if the activity in protocol record *j* is on the critical path of the work in work 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 work record *i*. The sub procedure is to determine further whether predecessors of the activity in protocol record *j* are on a critical path.

In the sub procedure, temporary set 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 may be on a critical path.

Main Procedure (*i*)
```
Step 1:  $j \leftarrow$ ProtocolPointer($i$) (protocol record $j$ keeps the current activity of
         the work kept in work record $i$);
Step 2:  OnCritical($p$) $\leftarrow$ FALSE, with that protocol record $p$ belongs to the
         work protocol till 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
         may be one of the last arrived), let
             LastArrivals ← LastArrivals ∪ {p}
Step 8:  if ArrivalTime(p) > LastArrivalTime (the activity in protocol record p
         may be the last arrived), let
             LastArrivalTime ← ArrivalTime(p) and
             LastArrivals ← {p}
Step 9:  go to Step 5;
Step 10: (all the activities in protocol records in set LastArrivals are on the
         critical paths) RestRecords ← LastArrivals;
Step 11: if RestRecords = ϕ, 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

In order to simulate how a process definition will be activated in the Espresso WfMS, the following input variables should be specified to a process definition:

- the life period of the process definition implemented in an application database;
- the rules for creating process instance in accordance with the process definition;
- the value distribution of a variable defined in a routing condition;
- the empirical distribution for "Exclusive Choice" outgoing links of an activity;
- the probability for routing a work along a "Multiple Choice" link;
- allowance for stopping a process instance; and
- property of a condition variable: activity-dependent and/or multi-values.

During a simulation run, routing work items are animated on a process map. Beside the icon of each activity, the number of the last treated process instance, the total number of running activity instances, the total number of created activity instances, and the 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 and algorithms relevant to parallel running work items have been 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 activity definition. 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 model are specified in a PAVONE Organization Database and/or a Notes Organization Directory, where human and material resources are already defined. For simulating the utilization of the resources, some resource-relevant assumptions are implemented in the simulation model.

During a simulation run, 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 if the editor assigned to an activity cannot 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. All of them can be simulated in PM.

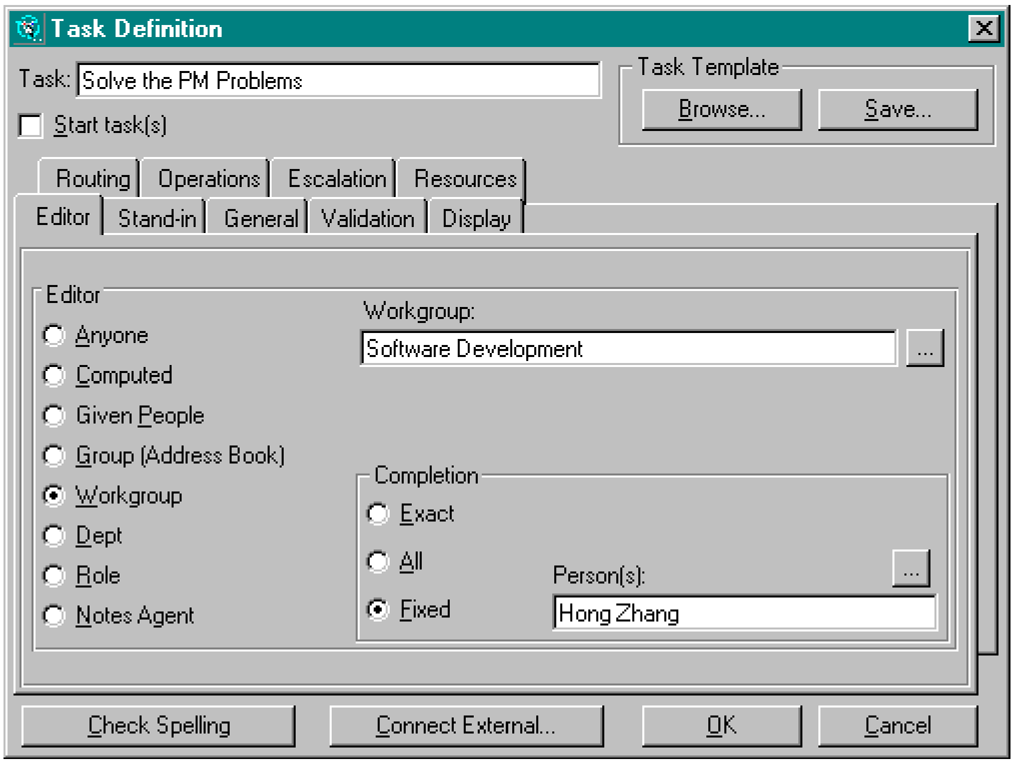
- *Human resource*: a person defined in a PAVONE Organization Database or a Notes Organization Directory. A person can be a member of a team defined in the databases.
- *Notes agent* (IT resource): a procedure of actions such as filling documents, sending mail, looking for particular topics, archiving older documents, manipulating field values, bringing data in from other applications, etc., on the pre-selected set of documents in an Espresso 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/Gabathuler/Jansen /Rossini/Wylie/Schaper, 1998, p. 22]
- *Material resource*: a non-shared machine, device, tool or room defined in a PAVONE 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. They 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., can be simulated and calculated.

12.1.1 Workflow Participant Specification

In the Espresso WfMS, workflow participants who can undertake an activity are assigned by one of the eight kinds of *editor*s via the PM dialogue box shown in Figure 12-1.



Task: Solve the PM Problems

☐ Start task(s)

Editor: Anyone / Computed / Given People / Group (Address Book) / [✓ Workgroup] / Dept / Role / Notes Agent

Workgroup: Software Development

Completion: Exact / All / [✓ Fixed]

Person(s): Hong Zhang

Figure 12-1. Activity Participant Definition

- *Anyone*: any person defined in a PAVONE Organization Database and/or a Notes Organization Directory.
- *Computed*: dynamic organizational role read from the given Notes field in the document representing an activity instance. The value of field can be computed at run-time.
- *Given people*: a number of people defined in a PAVONE 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 a PAVONE Organization Database.
- *Department*: an organizational role defined in a PAVONE Organization Database, with the specification of either including or excluding sub-departments.
- *Role*: an organizational role defined in a PAVONE Organization Database, with the specification of either a fixed parameter or a dynamic parameter reading at run-time from the given Notes field.
- *Notes agent*: an IT resource defined in an Espresso application database and able to automatically execute a 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,

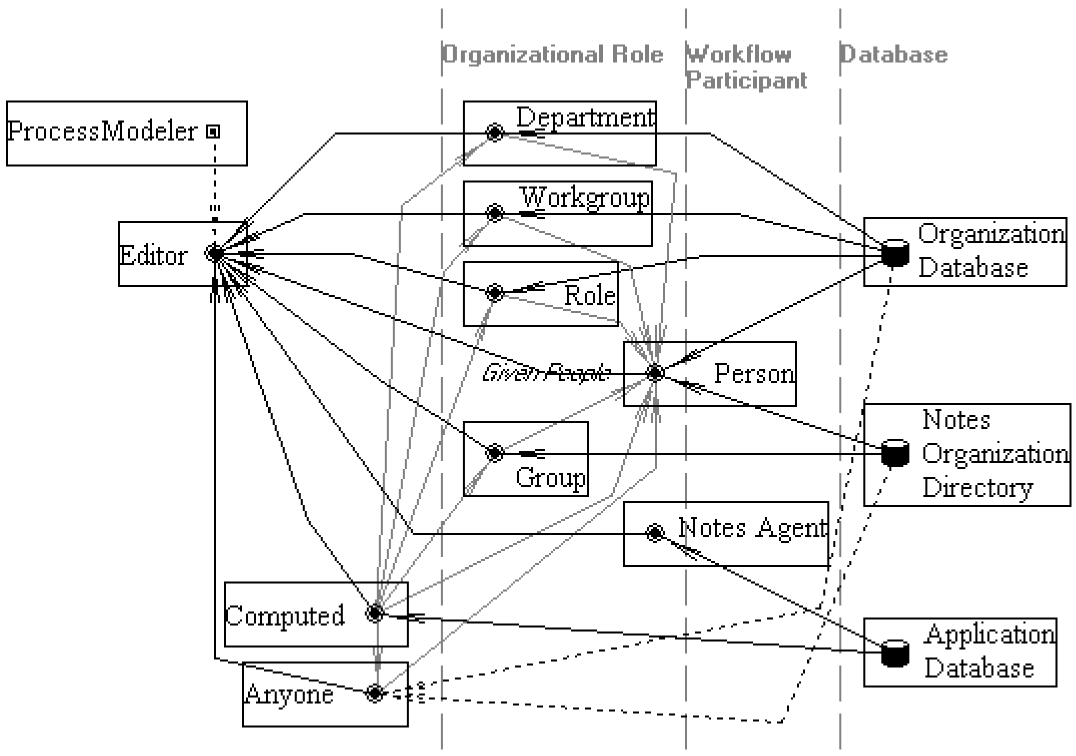


Figure 12-2. Editor, Organizational Roles and Workflow Participants

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 at run-time with a combination of several organizational roles. 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 the system environment, 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 to 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 is defined in the PAVONE Organization Database or what is given for the activity. The given stand-in for an activity is specified in the same way as an editor.

The stand-in belongs to exception handling and so it will not be simulated, just as ad-hoc workflows within a process instance are not simulated.

Assumption 12-2. No Simulation of a Stand-in
The stand-in of an activity is not simulated.

12.1.1.1 Simulating Editor "Computed"

In the simulation model it is disregarded that a Notes field specified for editor "Computed" could 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.

Assumption 12-3. Simulating Editor "Computed"
A number of people are directly resolved from editor "Computed" to the workflow participants of an activity.

Initiator of a process instance is the workflow participant who creates in an Espresso application database 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.

When a process creation event at a start activity of a process definition is scheduled, one of the workflow participants resolved from the editor assigned to the start activity is determined as the initiator of the process instance. For a simulation study, if the start activity can be executed only by the initiator, the activity should be assigned with editor "Computed" and field "wfInitiator", instead of with editor "Anyone".

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" that is assigned to an activity. The workflow participants of the activity are thus all those workflow participants who have executed the predecessor activities.

Assumption 12-4. Simulating Workflow Participants of the Predecessors
The workflow participants of an activity assigned to editor "Computed" with field "wfPrevMember" are resolved to 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 as workflow participants. Editor "Computed" has a field able to be filled at run-time with one or more of the teams. These editors are therefore called *team editor*s.

12.1.1.2.1 Activity Completion Specification

In the Espresso WfMS, if an activity is assigned to a team editor, the workflow engine will allocate a work associated with the activity to the worklists of all the team members, so any team member can execute the activity. Thus, it is necessary to specify through one of the three options in order to determine which or how many members of a team editor must complete an activity (see Figure 12-1).

1. *Exact*: specified number of team members must complete the activity.
2. *All*: all members of the team must complete the activity.
3. *Given*: specified members of the team must complete the activity.

To complete an activity within a process instance is for a workflow participant to issue in the Espresso application database a command (via a button or a menu) of completing the activity, and then routing the work of the associated process instance further to some successor activities or terminating the process instance if the activity is an end activity.

In the real world WfMS, if an activity is assigned to a team editor, team members who execute the activity and a member who completes the activity may undertake the activity in different ways. But in the Espresso 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, delay handling, 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 an exact number of members of a team editor is specified for the activity completion, the members who must complete the activity are uncertain.

12.1.1.2.2 Simulating Work Allocation among a Team

In the Espresso WfMS, in addition to the team members who must complete an activity associated with a process instance, other team members can execute the activity too. However this is not always actually allowed. Therefore, it can be specified for the simulation study, whether just team members who must complete the activity can execute the activity or not (see Figure 12-3 in Section 12.1.2).

In the Espresso 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 a PAVONE Organization Database or a Notes Organization Directory (see Section 12.3.3). A work can also be allocated to the dummy participant of a team editor (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 must complete the activity, the workflow participants to execute the activity are uncertain and will be determined in the following steps. Suppose that N is the given exact number of members or the number of given members for completing the activity, and n is the number of team members able to participate in the work associated with the process definition.

```
1°  Determine $m$, the number of workflow participants:
    if just the specified members for completing activity can be workflow
    participants,
        $m$ is determined as the same as $N$;
    otherwise (every team member can be a workflow participant),
        $m$ is determined by generating a variate governed by $U[1, n]$
        distribution, and then is assigned with $N$ if determined $m$ is less than $N$
        (because the members specified for completing the activity must be
```

workflow participants);

2° Determine *m* workflow participants among the team:
if given exact number of members for completing 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 completing the activity),
the given *N* 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 the members of a team;

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 in order to execute the activity); or
- if some of the determined workflow participants (e.g. someone specified in the distribution or given for completing the activity) will be removed since they do not participate in the work associated with the process definition or are not simulated.

From this assumption, if not all team members are defined to 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 should not be less than the given exact number of members for completing the activity.

12.1.1.3 Simulating the Worklist of a Workflow Participant

"The queue is a set of jobs that are waiting for service." [Fishman, 1973, p. 31] The worklist of a workflow participant in a WfMS are simulated with a waiting queue. So the queuing rule for the simulation study should be specified (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 be considered;
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 cannot switch

membership to an alternate waiting queue that might, for example, have become short.

Unlike in the real world Espresso WfMS where a work is allocated to all team members resolved from a team editor by the 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 of the Espresso WfMS, since 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 the activity. Activity execution time refers to the time taken by the workflow participants to execute the activity. Therefore, 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.

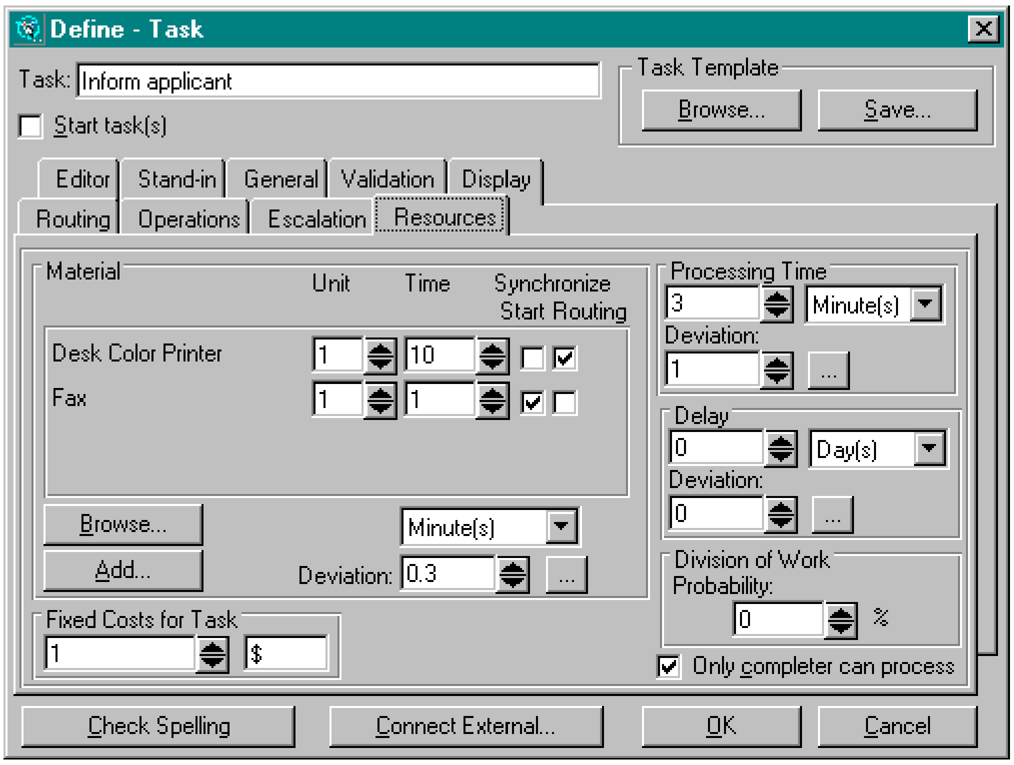
The elapsed time period that is specified to an activity to just prolong the duration of the activity is called a *delay* of the activity. During delay of an activity, the workflow participants do nothing for the activity. Thus, the delay time is not taken into account in the calculation of the costs of human resources. For example, before a workflow participant can execute an activity, he must call somebody outside the WfMS for gathering 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 cannot execute the activity, but he can execute other activities in his worklist. Since the following factors have been considered in the simulation model for influencing the duration of an activity, they should not be included in the specified delay time of an activity:

- enlarged execution/occupation time of a resource because of part-time engagement,
- 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.

The execution time (processing time) and delay time of an activity (task) are specified via the dialogue box of PM as shown in Figure 12-3. The execution



Task: Inform applicant

☐ Start task(s)

Material

Material	Unit	Time	Synchronize Start	Synchronize Routing
Desk Color Printer	1	10	☐	☑
Fax	1	1	☑	☐

Minute(s)

Deviation: 0.3

Fixed Costs for Task: 1 $

Processing Time: 3 Minute(s)
Deviation: 1

Delay: 0 Day(s)
Deviation: 0

Division of Work
Probability: 0 %

☑ Only completer can process

Figure 12-3. Activity Resource Definition

time and the delay time given here 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 PAVONE Organization Database (see Section 12.3.1).

Assumption 12-9. Distribution of Execution/Delay Time of an Activity
The execution time and delay time of an activity in an organization follow the same distribution function with different means and standard deviations. The means and standard deviations are individually specified to each activity within a process definition.

The standard deviations specified respectively for the execution and 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.

The simulator 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 a real world WfMS, it may happen that a workflow participant interrupts executing an activity in order to execute another one. But in the simulation model, this is not allowed.

Assumption 12-10. Continuous Activity Execution
During a simulation run, once a workflow participant executes an activity, he continues executing it and cannot execute other activities before finishing the execution of the current activity.

In a real world WfMS, delay may happen one or more times during execution of an activity. But in the simulation model there is the following assumption.

Assumption 12-11. Simulating Delay of an Activity
When a workflow participant prepares to execute a work associated with an activity, the delay of the execution will begin, if the determined delay time is larger than zero. The delay happens only before execution of the activity. During the delay, the work 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 when he becomes no longer busy and the delay time has been elapsed.

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 cannot be undertaken. A material defined for an activity must have 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 cannot be used for execution of another activity.

A printer or a fax device, for example, is a material that can be used for execution of an activity. In the PM dialogue box shown in Figure 12-3, the materials as well as the number of units required for the execution of the activity (task) can be specified. Furthermore, synchronization of each material can be specified through occupation time, synchronous start and synchronous routing.

- If a material is start-synchronous, a workflow participant cannot execute the activity until the material is available to be used for the activity.
- If a material is routing-synchronous, after completion of the activity, the work associated with the activity cannot be routed or terminated till the material is released from the execution of the activity.
- 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. In this case, the material is both start- and routing-synchronous.

For example, a letter scanned at an activity within a process instance must be routed with the work associated with the process instance to the next activity, and a scanner should, thus, be specified as a routing-synchronous material for the activity. A projector should be both start- and routing-synchronous, if it must be used during a presentation, an activity within a process instance, and therefore the occupation time for the projector should not be specified—the occupation time of the material is the same as the execution time of the activity.

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 to a PAVONE 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 as a workflow participant cannot be interrupted during execution of an activity, the use of a material resource cannot 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 cannot be used for execution of any other activities before it is released from the occupation.

An activity cannot 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 it is routing-synchronous. Thus, shortages of synchronous material resources will prolong the duration of an activity.

In the PAVONE Organization Database, the maximum number of units for a material can be defined as one or more. Therefore, it is also allowed to let multiple units of the material to be used for the execution of an activity. In the simulation model, it is assumed that all the work items demanding the use of a material are waiting in one queue, ignoring whether the maximum number of units of a material is one or more.

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 concerned during a simulation run. Therefore, 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 that can achieve organization objectives with the best service level or efficiency and the least costs. With help of the simulation study, the 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 PAVONE Organization Database configured to the Espresso application database, in which the process instances of a process definition are simulated.

Apart from the resource costs calculated upon the hourly 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 that 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 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 simply reducing human resources may prolong the duration of the activities and hence the overall duration of a process instance.

12.1.5 Simulating Multiple Workflow Participants

It is known that a work associated with an activity can be allocated to multiple workflow participants that are resolved from a team editor assigned to the activity. The work is then a team work.

In the real world Espresso WfMS, an activity instance is represented by a Notes document. If the work associated with the activity instance is assigned to a team editor, it can be executed by all the team members and the document is shared among them. To avoid replication conflict, the document can be locked when one member is editing it, so that none of the others 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 a work associated with an activity is allocated to multiple members of a team as workflow participants, at what time and how long a workflow participant executes the activity are independent on one another.

Here 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.

Assumptions 12-15 and 12-16 are not practical, when a team work (e.g. a meeting) associated with an activity must be executed by the workflow participants at the same time with just one unit of a material (e.g. a project) for the execution of the activity (see Section 13.6).

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 or not will be determined according to the given probability (i.e. a 0-1 distribution).

Assumption 12-17. Dividing Work among Multiple Workflow Participants
Suppose that μ and σ 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 μ / 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 μ and σ respectively.

In Assumption 12-17, the formula for dividing time is based upon the central limit theorem.

It is obvious from this assumption that for a work dividable activity, such an activity can be executed through cooperation by multiple people; the more workflow participants work together, the shorter the potential duration 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 simulated person. In this case, the dummy participant of the editor is simulated. The dummy participant of an editor is used where one or more of the following conditions apply.

- Neither the PAVONE Organization Database nor the Notes Organization Directory is configured to the Espresso application database, where the process instances of a process definition are simulated—except for editor “Given people”, any other editors defined in the process definition get a dummy participant.
- Editor “Anyone” or editor “Computed” is assigned to an activity—the dummy participant stands for the people in the configured PAVONE Organization Database and/or Notes Organization Directory who have no specification about participation in the work associated with the process definition, and cannot be resolved from a team editor either.
- A team organizational role (i.e. department, workgroup, role or Notes group) is not defined in the configured PAVONE 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 must 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 people to complete an activity are excluded from the simulation study.
- The human resources participating in a work associated with the process definition are excluded from the simulation study.

The dummy participant of an editor assigned to an activity within a process definition is named by “(Dummy: <Team Name>)”, such as “(Dummy: Anyone)”, supplemented with a combination of ReplicaIDs of a PAVONE Organization Database, a Notes Organization Directory, and/or an Espresso 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 the process instances of a process definition are 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 resource integrity errors within the process definitions.

Assumption 12-18. Dummy Participant of an Editor

- A 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. That is, 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 for the standard weekly working hours specified for the simulation study.

Except those mentioned in Assumption 12-18, 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 work is allocated to a dummy participant, it can be executed 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 statistic data can be simply analyzed.

12.3 Organizational Settings

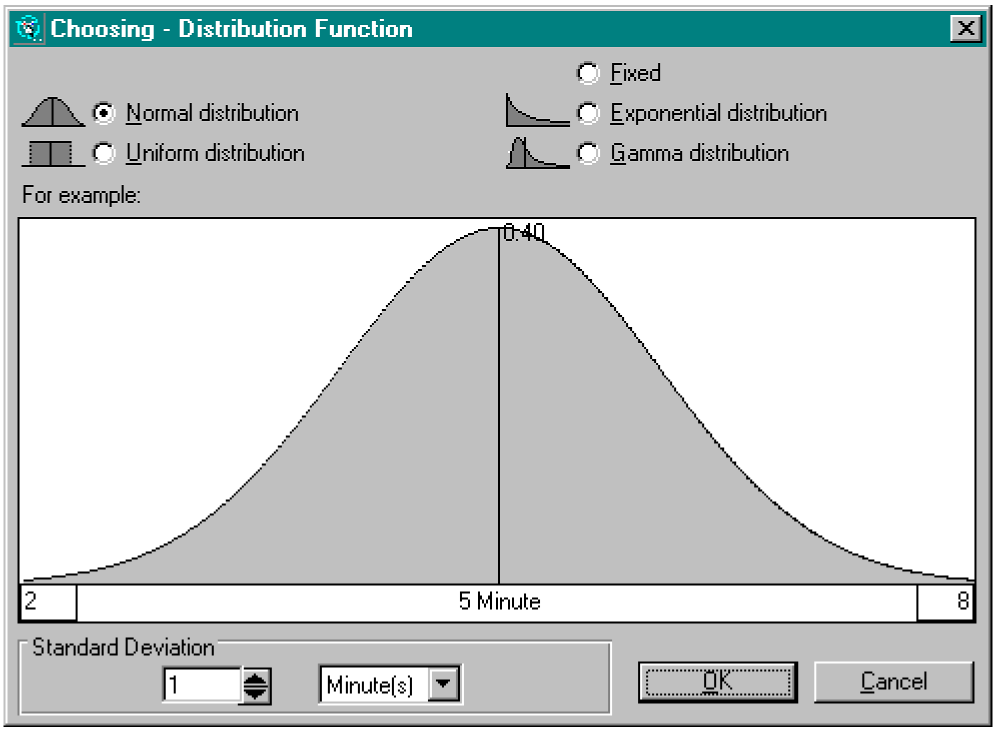
Human and material resources are defined in the PAVONE Organization Databases and Notes Organization Directories. The simulation settings specified to the databases are organizational settings. They can be categorized into four groups:

- 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 process definition configured with the PAVONE Organization Database or the Notes Organization Directory, if there are no corresponding process-dependent organizational settings for the analyzed process definition.

12.3.1 Distribution Function Settings

The distribution functions are specified for the simulator to generate activity execution times as well as delay times, work routing times and material occupation times. Parameters of the distribution functions (i.e. mean and standard deviation) are specified with the definitions of an activity or a link. The distribution functions can be specified as either normal, uniform, exponential or gamma distribution, as shown in Figure 12-4.


Choosing - Distribution Function

[✓ Normal distribution]
Uniform distribution
Fixed
Exponential distribution
Gamma distribution

For example:

Standard Deviation: 1 Minute(s)

Figure 12-4. Distribution Function Specification

The curve of a distribution function with an example of the 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 probable it is 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 in the dialogue box in Figure 12-4 is particularly useful when the system analyst want to test and validate the simulator, to comprehend the simulated system, or to compare alternative operating policies.

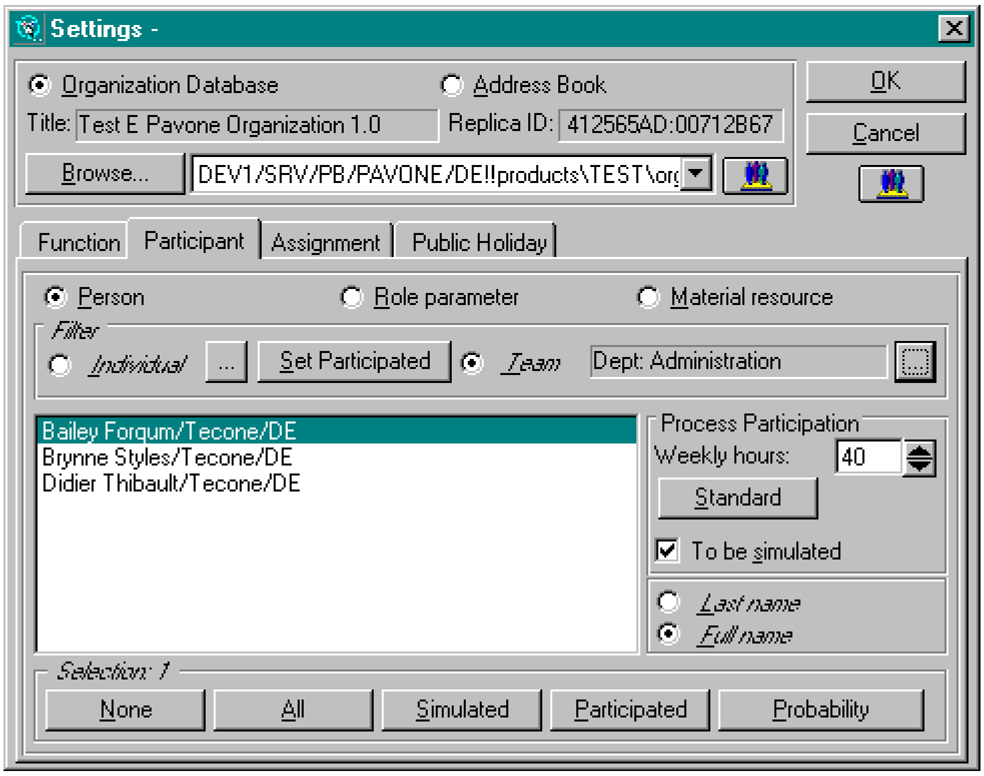
Assumption 12-19. Effect of Zero Standard Deviation
If the standard deviation is specified with zero, the generated time will always be the same as the mean, no matter which distribution function the time follows.

12.3.2 Participant Settings

Many 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.
- It is complicated to analyze the simulation results involved with many resources and so difficult to make decisions upon them.

The dialogue box shown in Figure 12-5 is used to specify weekly hours that a human or material resource can participate in a work associated with a



[✓ Organization Database] Address Book
Title: Test E Pavone Organization 1.0
Replica ID: 412565AD:00712B67
DEV1/SRV/PB/PAVONE/DE!!products\TEST\or
[✓ Person] Role parameter Material resource
Filter: Individual [✓ Team] Dept: Administration
Bailey Forqum/Tecone/DE
Brynne Styles/Tecone/DE
Didier Thibault/Tecone/DE
Process Participation
Weekly hours: 40
[✓ To be simulated]
Last name [✓ Full name]
Selection: 1

Figure 12-5. Participant Specification

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 a 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 cannot have zero weekly participating hours for a process definition.

If a member of a team editor has in the Espresso Simulation Database no specification about the participation in the work associated with a process definition, as a default, 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 can be simulated. If a team editor is assigned to an activity within a process definition and all team members must complete the activity, the work belonging to a member who is not participating in the work associated with the process definition will be allocated to the dummy participant of the editor. Suppose that a role has ten members but only two of them participate in the work associated with a process definition, during a simulation run, 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 must complete the activity.

The non-zero weekly participating hours of a resource and the standard weekly working hours are used together to determine the duration of an activity.

Assumption 12-21. Work Calendar
The standard weekly working hours is specified in the calendar settings of the user preferences of PM, via weekly working days and daily working hours.

According to the assumption, all organizations involved in a simulation run have the same standard weekly working hours.

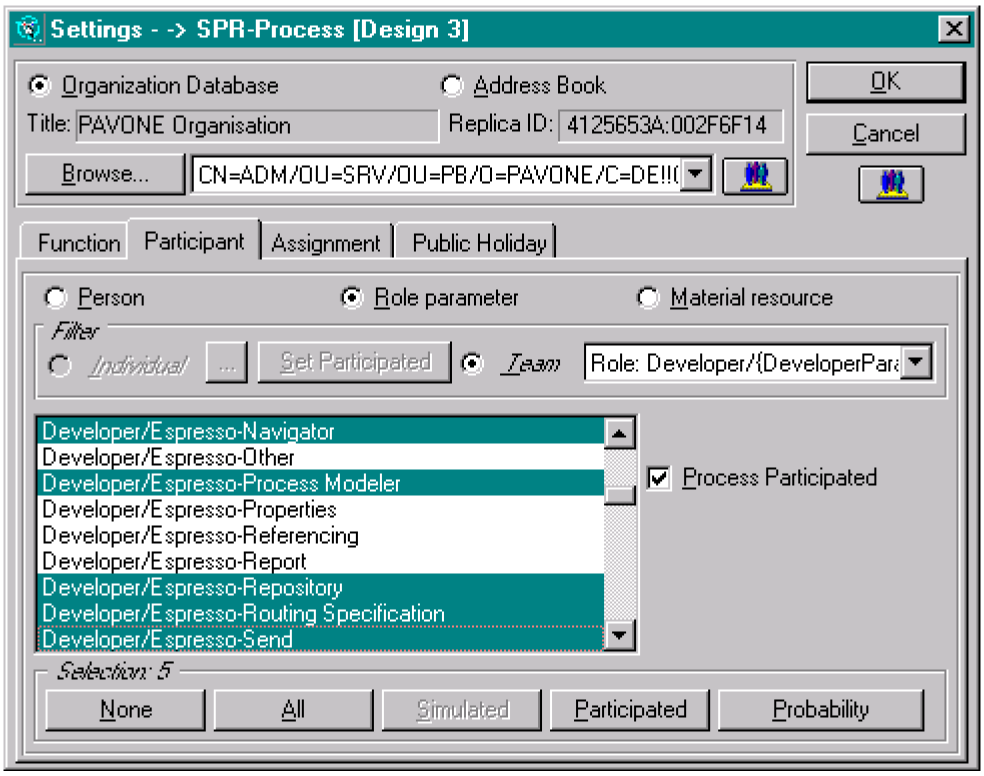
Assumption 12-22. Effect of Participant Settings
The duration caused by a human or material resource is calculated by

$$t\,(S/P)$$

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 or for using the material.

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 value of the weekly participating hours of a person, for example, is larger than the value of the standard weekly working hours, he works overtime and so needs less time to execute an activity. Suppose that 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 of the activity executed by this person will be, on average, 200 hours.

A team having a specified parameter or having no parameter, such as a workgroup, must participate 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 sub teams with specific parameters may actually not participate in the work associated with a process definition. This can be set in the dialogue box as shown in Figure 12-6.


Settings - -> SPR-Process [Design 3]

[✓ Organization Database] Address Book

Title: PAVONE Organisation

Replica ID: 4125653A:002F6F14

Browse... CN=ADM/OU=SRV/OU=PB/O=PAVONE/C=DE!!(

Function | Participant | Assignment | Public Holiday

Person [✓ Role parameter] Material resource

Filter: Individual ... Set Participated [✓ Team] Role: Developer/{DeveloperPar

Developer/Espresso-Navigator ✓
Developer/Espresso-Other
Developer/Espresso-Process Modeler ✓
Developer/Espresso-Properties
Developer/Espresso-Referencing
Developer/Espresso-Report
Developer/Espresso-Repository ✓
Developer/Espresso-Routing Specification ✓
Developer/Espresso-Send ✓

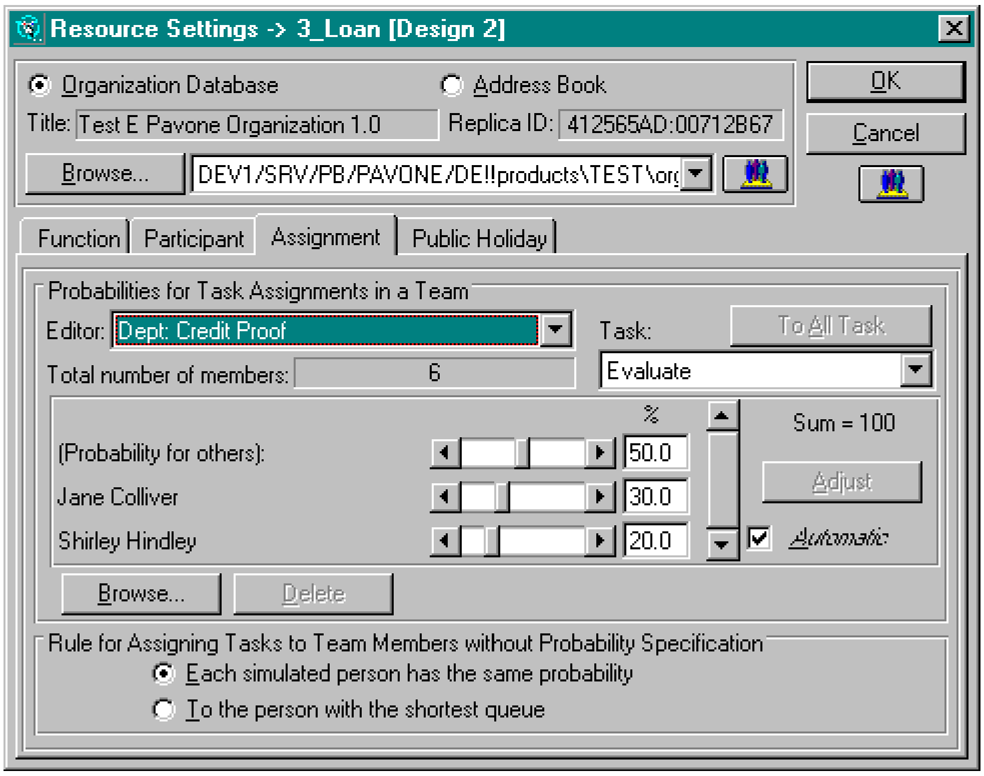
[✓ Process Participated]

Selection: 5

Figure 12-6. Participating Settings of Team Parameters

12.3.3 Team Work Assignment Settings

The empirical distribution for allocating a work among members of a team can be specified in the dialogue box shown in Figure 12-7. Once it is associated with a process definition, it is activity-dependent.


Resource Settings -> 3_Loan [Design 2]

[✓ Organization Database] Address Book

Title: Test E Pavone Organization 1.0 Replica ID: 412565AD:00712B67

DEV1/SRV/PB/PAVONE/DE!!products\TEST\or

Probabilities for Task Assignments in a Team

Editor: Dept: Credit Proof

Task: Evaluate

Total number of members: 6

	%
(Probability for others):	50.0
Jane Colliver	30.0
Shirley Hindley	20.0

Sum = 100

✓ Automatic

Rule for Assigning Tasks to Team Members without Probability Specification

[✓ Each simulated person has the same probability]

To the person with the shortest queue

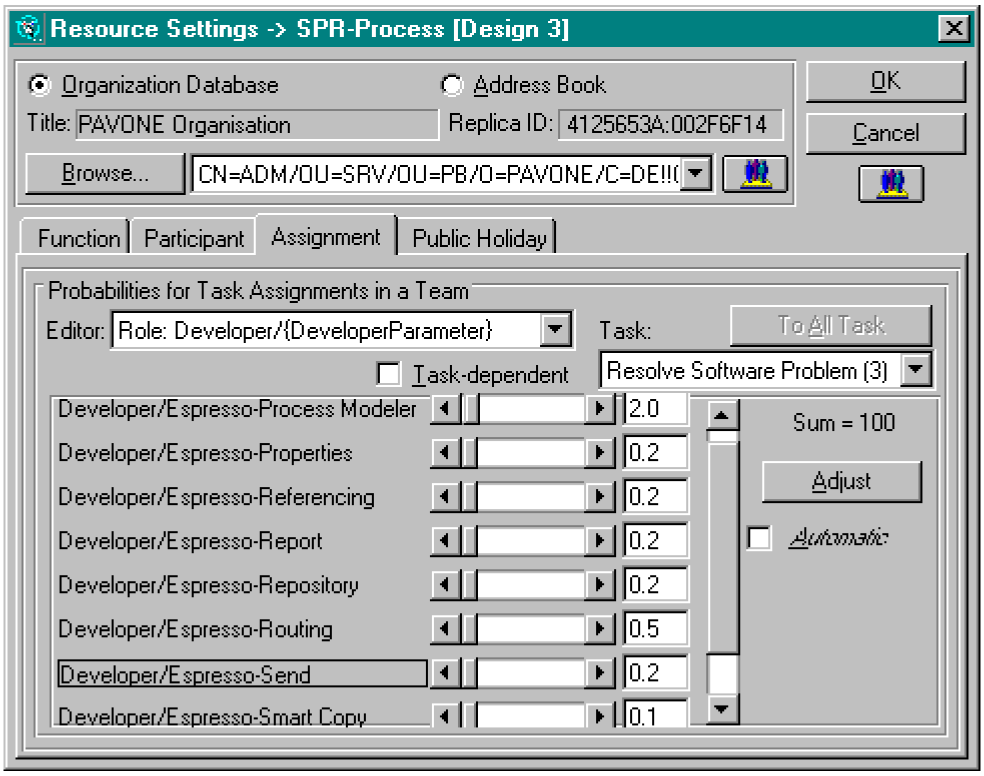
Figure 12-7. Assignment Specification

Not all team members must be specified in the empirical distribution. The probability for others is the sum of the probabilities of the 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, according to the specification by the system analyst in this dialogue box.

For 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 must complete it, the probability for Jane Colliver, Shirley Hindley and one of the other four members to execute the work are 30.0%, 20.0% and 12.5% (= 50.0% /4) respectively.

According to the work allocation distribution for an activity (task) within a process definition, the simulator will allocate a work associated with the activity to 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 in the distribution is larger than zero. 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 resolving workflow participants is given at run-time, the allocation distribution for the simulator to determine the parameter of the team can also be specified as shown in Figure


Resource Settings -> SPR-Process [Design 3]

[✓ Organization Database] Address Book

Title: PAVONE Organisation

Replica ID: 4125653A:002F6F14

CN=ADM/OU=SRV/OU=PB/O=PAVONE/C=DE!!(

Probabilities for Task Assignments in a Team

Editor: Role: Developer/{DeveloperParameter}

Task: Resolve Software Problem (3)

☐ Task-dependent

Developer/Espresso-Process Modeler	2.0
Developer/Espresso-Properties	0.2
Developer/Espresso-Referencing	0.2
Developer/Espresso-Report	0.2
Developer/Espresso-Repository	0.2
Developer/Espresso-Routing	0.5
Developer/Espresso-Send	0.2
Developer/Espresso-Smart Copy	0.1

Sum = 100

☐ Automatic

Figure 12-8. Assignment Specification—Team Parameters

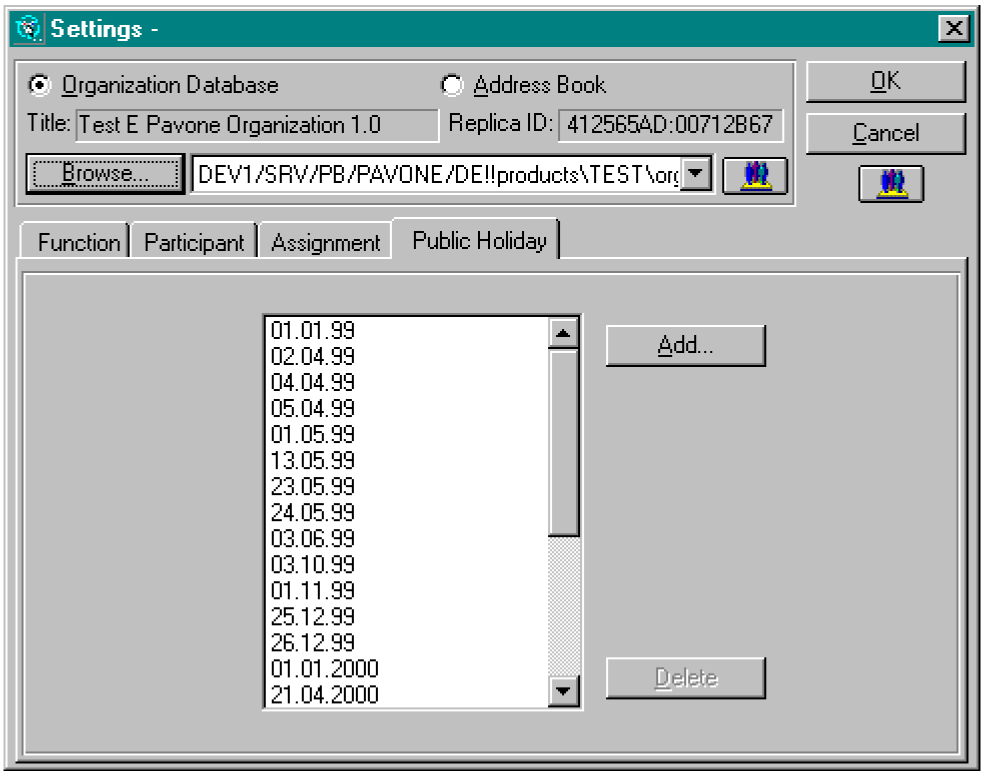
12-8. In the allocation distribution, all the parameters of a team will be automatically filled in the list.

12.3.4 Public Holiday Settings

Public holidays of an organization are specified in the dialogue box shown in Figure 12-9. The public holidays of the database can be 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 with a date field and 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 simulator transforms the value of the clock into a date, or from a large time unit (such as week) into a small unit (such as hour). They are also excluded during date specification via 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 system analyst wants the state and statistic data to be related with the date during a simulation run, he should specify the simulation beginning date.



(● Organization Database) ○ Address Book

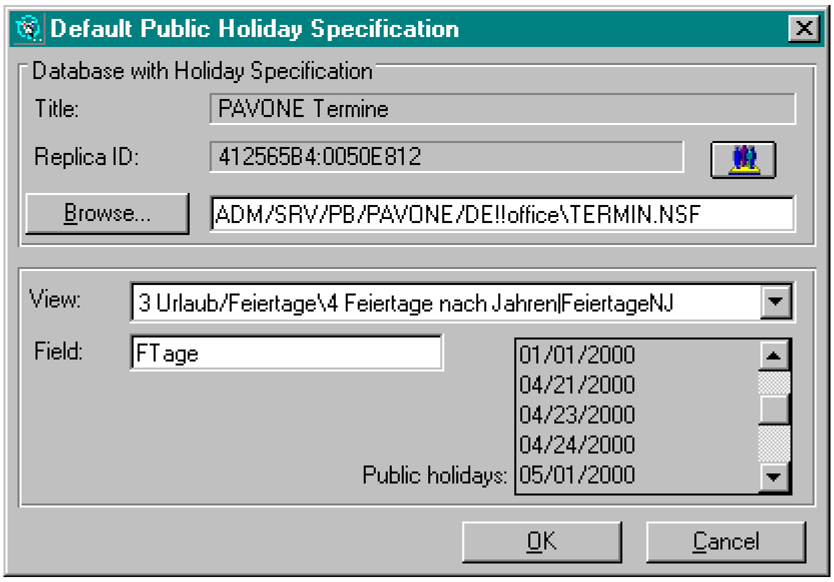
Title: Test E Pavone Organization 1.0

Replica ID: 412565AD:00712B67

DEV1/SRV/PB/PAVONE/DE!!products\TEST\or

01.01.99
02.04.99
04.04.99
05.04.99
01.05.99
13.05.99
23.05.99
24.05.99
03.06.99
03.10.99
01.11.99
25.12.99
26.12.99
01.01.2000
21.04.2000

Figure 12-9. Public Holiday Specification


Default Public Holiday Specification
Database with Holiday Specification
Browse...
OK
Cancel

Title: PAVONE Termine

Replica ID: 412565B4:0050E812

ADM/SRV/PB/PAVONE/DE!!office\TERMIN.NSF

View: 3 Urlaub/Feiertage\4 Feiertage nach Jahren|FeiertageNJ

Field: FTage

Public holidays:
01/01/2000
04/21/2000
04/23/2000
04/24/2000
05/01/2000

Figure 12-10. Default Public Holiday Specification

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 a PAVONE Organization Database and a Notes Organization Directory are configured to an Espresso application database where process instances of a process definition will be simulated, only the common public holidays that are defined both in the organization database and in the Notes Organization Directory are considered by the simulator.

Assumption 12-23. Common Public Holidays
If process instances of one or several process definitions will be simulated in diverse Espresso application databases, only the holidays included in all configured PAVONE Organization Databases and Notes Organization Directories will be considered in transforming the absolute simulation time into a date.

For example, if March 8 is a public holiday in the PAVONE Organization Database, but not in the Notes Organization Directory, the day is not treated as a public holiday of the simulated WfMS.

12.4 Graphic States of Resources

Human resources, material resources and Notes agents can be assigned to activities within a process definition. In the Espresso WfMS, the human resources will be obtained from either a Notes Organization Directory or a PAVONE Organization Database, or both; material resources will be retrieved from the organization database; and Notes agents are programmed in the Espresso application database. Because some resources defined in the relevant databases are not used in a process definition, they will not be considered in the simulation study.

The system analyst can specify for the simulation study whether a human or material resource is allowed to participate in the work associated with a process definition. Resources participating in a work associated with the process definition must have a non-zero weekly participating hours. Only the resources participating in the work associated with the analyzed 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.

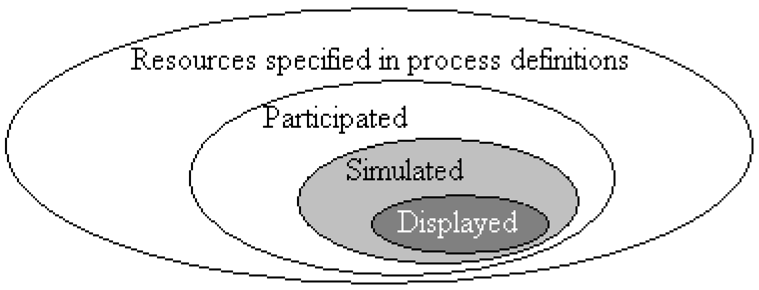


Figure 12-11. Simulated Resources

Every editor 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.

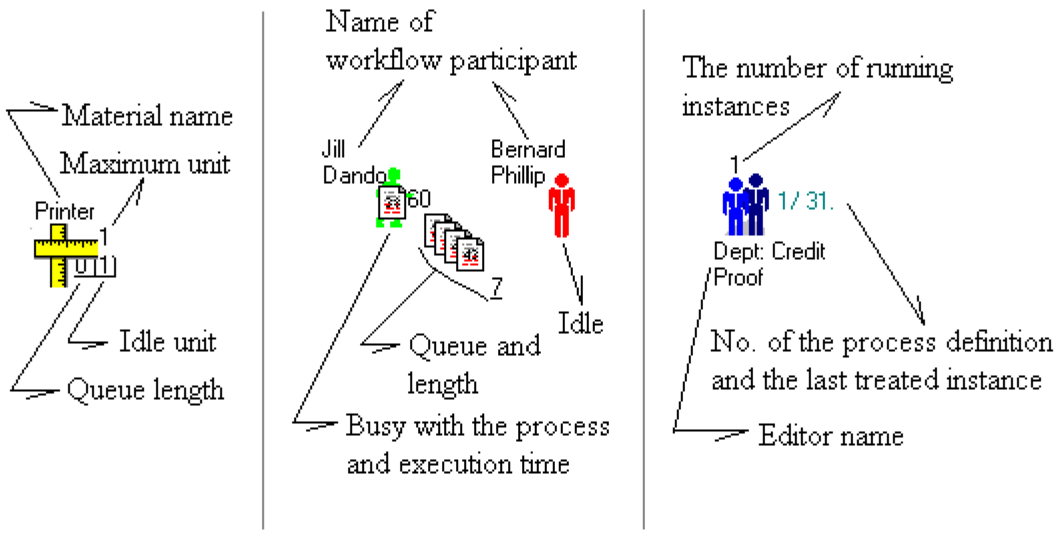

Material name
Maximum unit
Printer
1
0
1
Idle unit
Queue length
Name of workflow participant
Jill Dando
60
Bernard Phillip
7
Queue and length
Idle
Busy with the process and execution time
The number of running instances
1
1/ 31.
Dept: Credit Proof
No. of the process definition and the last treated instance
Editor name

Figure 12-12. Graphic Resource States

- 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 sequence of document icons may appear. If a workflow participant is busy with a work associated with a process instance, the document icon with the number of the process instance and the time needed by him to undertake the work are displayed on the icon of the workflow participant.
- Editor: the icon of a pair of people represents the simulated 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 of the first process definition opened in PM, is the last one to be routed.

In the application database of the Espresso WfMS, 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 Espresso simulation model. 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 modified 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 document icons displaying in a queue.

A document icon in the resource window represents a work associated with a simulated process instance in accordance with a process definition. The number of the process instance is displayed on it. In addition to that a document icon can be used to build up a queue, it is also utilized to animate a flowing work. During a simulation run, 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 the simulator allocates a work associated with the activity which is 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 is just flowing. 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 Suellyn Hayes and Peter Marley. Meanwhile, 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 routed from editor "Department" of "Credit Proof" to editor "Department" of "Asset Value Proof".

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 having the potential to flow to the join activity.

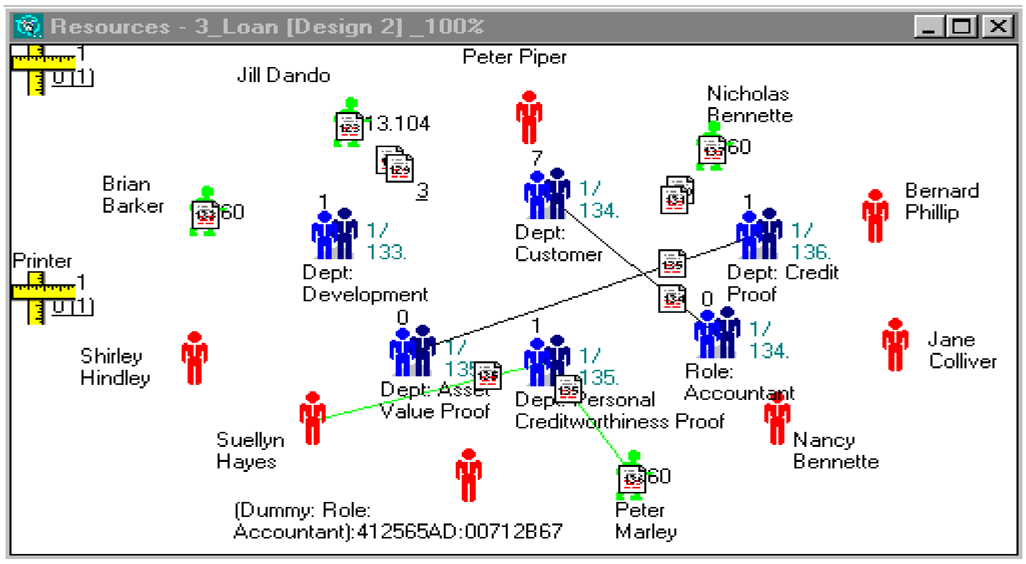


Figure 12-13. Dynamic Resource Window

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 of 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 used as an auxiliary tool to detect potential resource bottlenecks, via observing a simulation run. For example, if a high number 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

In the simulation model, there is a waiting queue of work items for each simulated human workflow participant who is kept in a participant record. The queue records combining the waiting queue are connected by two pointers of each queue record as shown in Figure 12-14—one is pointed to the next queue record and another to the previous queue record. In the participant record, there

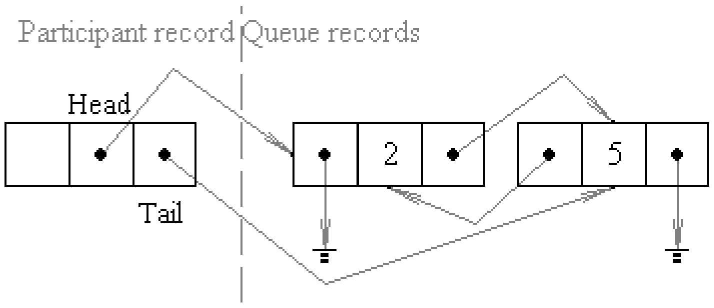


Figure 12-14. Queue Records

are two pointers used to connect respectively the head and tail of the waiting 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 of Figure 12-14, there are two work items waiting in the queue, one needs 2 minutes to execute and the other 5 minutes. With this data structure, it is easy to handle a queue with the queuing rules of first-in-first-out or last-in-first-out.

Now suppose that 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.

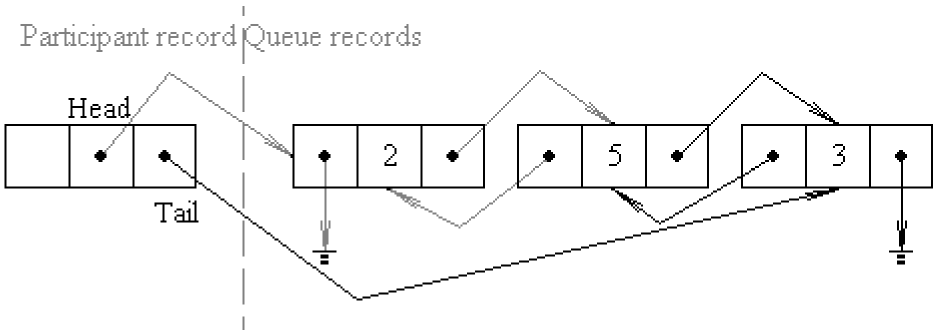


Figure 12-15. Queue Records—First-in-First-out

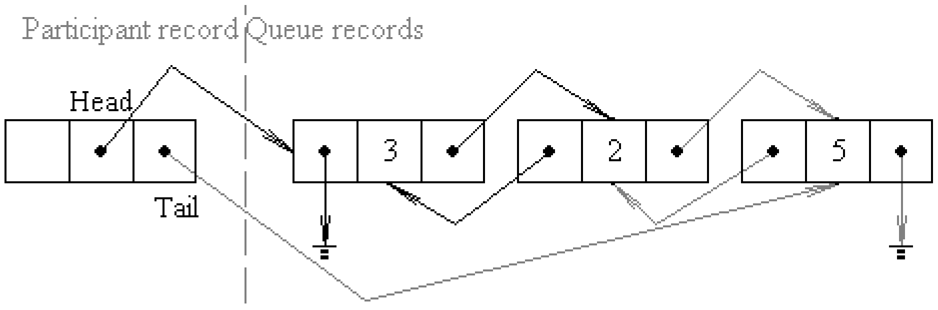


Figure 12-16. Queue Records—Last-in-First-out

If a queue is ordered in respect to execution time of a work item, a tree structure (see Section 8.2.2.3) is utilized for sorting the queue records. In this case, the head pointer of a participant record connects to the tree root of the waiting queue and the tail pointer does not needed.

12.5.1 Algorithm for Adding Work to Queue

This procedure is called when processing a work allocation event (see Section 13.4) in order to add a work in the queue of a workflow participant. If the queuing rules are first-in-first-out or last-in-first-out, the data structure discussed here is used for inserting a new queue record; otherwise the tree structure like that discussed in Section 8.2.2.3 is used. See Section 8.2.2.3.1 for a similar algorithm for inserting a new queue record into a tree sorted in respect to the execution time.

Hypothesis

QueueLength(*p*), Head(*p*) and Tail(*p*) represent respectively queue length, the queue head pointer and the queue tail pointer of the workflow participant kept in participant record *p*. ExecutionTime(*j*), Next(*j*) and Previous(*j*) stand for respectively the execution time, the next queue record pointer (or the right pointer) and the previous queue record pointer (or the left pointer) of the work kept in queue record *j*.

Principle

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*, according to the given queuing rule (see Figure 12-14, Figure 12-15, Figure 12-16 and Figure 8-2).

Procedure (*p*, *i*)

```
Step 1: ExecutionTime(i) ← generating execution time for the work in queue
        record i;
Step 2: if QueueLength(p) > 0 (the queue of the workflow participant in
        participant record p is already existing), go to Step 4;
Step 3: if the queuing rule is first-in-first-out or last-in-first-out (to build up a
        new queue for the workflow participant kept in participant record p),
            1° Next(i) ← 0 and Previous(i) ← 0;
            2° Head(p) ← i and Tail(p) ← i;
            3° go to Step 9;
        otherwise let Head(p) ← 0 and go to Step 6;
Step 4: if the queuing rule is first-in-first-out, go to Step 7;
Step 5: if the queuing rule is last-in-first-out, go to Step 8;
Step 6: (the queue is ordered according to the least execution time) call the
        algorithm for inserting the new queue record into a tree sorted in
```

```
         respect to ExecutionTime() with parameters Head(p), Head(p), i and
         FALSE, and then go to Step 9;
Step 7:  (to append queue record i to the tail of the queue) Next(i) ← 0;
         Previous(i) ← Tail(p); Next(Tail(p)) ← i; Tail(p) ← i; go to Step 9;
Step 8:  (to insert queue record i to the head of the queue) Next(i) ← Head(p);
         Previous(i) ← 0; Previous(Head(p)) ← i; Head(p) ← i;
Step 9:  accumulate the integral of the queue length for the workflow
         participant in record p (see Section 8.2.3.4);
Step 10: let QueueLength(p) ← 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) in order to remove the queue record which keeps the first executable work (all required material resources are available and delay is no more required) from the queue. It is called before a workflow participant begins executing an activity.

Hypothesis
QueueLength(*p*), Head(*p*) and Tail(*p*) represent the queue length, the queue head pointer and the queue tail pointer of the workflow participant in participant record *p*. Next(*j*) and Previous(*j*) represent the next queue record pointer (or the right pointer) and the previous queue record pointer (or the left pointer) of a work kept in queue record *j* respectively.

Principle
Here participant record *p* is a parameter of the procedure. This procedure removes the queue record of the first executable work from the queue belonging to the workflow participant in record *p*. The queue record will be returned by the procedure. If no executable work is removed from the queue, value 0 will be returned.

If the queuing rule is first-in-first-out or last-in-first-out, sub procedure1 will be called; otherwise sub procedure2 will be called to get the queue record from a tree ordered in respect to execution time of a work item.

RootTree, CurRecord and FatherRecord are the parameters of the self-called sub procedure2. RootTree represents the root pointer of the tree. Record CurRecord in the tree is the current treated record and is connected by the left pointer or the right pointer of record FatherRecord. The self-called sub procedure3 with the parameters of queue records *p* and *s* will be called to connect record *s* after all the records connected by record *p* in the tree structure. Temporary variable NewConnectRecord is used in sub procedure2.

```
Procedure (p)
Step 1: if the queuing rule is first-in-first-out or last-in-first-out, let
            i ← call sub procedure1 with parameter p;
        otherwise, let
            i ← call sub procedure2 with parameters Head(p), Head(p) and 0;
Step 2: accumulate waiting time of the queue of the workflow participant in
        record p;
Step 3: accumulate the integral of the queue length for the workflow
        participant in participant record p;
Step 4: let QueueLength(p) ← QueueLength(p) – 1; stop (return i).

Sub Procedure1 (p)
Step 1: let i ← the queue record keeping the first executable work item in the
        queue of the workflow participant kept in record p; if i = 0, go to Step
        7;
Step 2: if i = Head(p), go to Step 5;
Step 3: if i = Tail(p), go to Step 6;
Step 4: (to remove queue record i from the middle of the queue)
        Next(Previous(i)) ← Next(i); Previous(Next(i)) ← Previous(i); go to
        Step 7;
Step 5: (to remove queue record i from the head of the queue) Head(p) ←
        Next(i); Previous(Next(i)) ← 0; go to Step 7;
Step 6: (to remove queue record i from the tail of the queue) Tail(p) ←
        Previous(i); Next(Previous(i)) ← 0;
Step 7: stop (return i).

Sub Procedure2 (RootTree, CurRecord, FatherRecord)
Step 1: if Previous(CurRecord) ≠ 0 (record CurRecord is not the first record in
        the tree), let i ← call the sub procedure2 self with parameters
        RootTree, Previous(CurRecord) and CurRecord;
Step 2: if i ≠ 0 (have got the first executable work kept in record i) go to Step
        12;
Step 3: (to consider the work in record CurRecord) if work in record
        CurRecord is not executable, go to Step 11;
Step 4: (work in record CurRecord is the first executable in the queue) let
            i ← CurRecord;
Step 5: (to disconnect record i from the tree) if Previous(CurRecord) = 0, let
            NewConnectRecord ← Next(CurRecord) and go to Step 8;
Step 6: let NewConnectRecord ← Previous(CurRecord);
Step 7: if Next (CurRecord) ≠ 0, call the sub procedure3 with parameters
        Previous(CurRecord) and Next (CurRecord);
Step 8: if FatherRecord ≠ 0, go to Step 10;
Step 9: (record CurRecord is connected by the root pointer of the tree), let
            RootTree ← NewConnectRecord and go to Step 12;
```

```
Step 10: (to change connection from record FatherRecord) if
         Previous(FatherRecord) = CurRecord (record CurRecord is connected
         by the left pointer of record FatherRecord) let
             Previous(FatherRecord) ← NewConnectRecord and go to Step 12;
         Otherwise (record CurRecord is connected by the right pointer of
         record FatherRecord) let
             Next(FatherRecord) ← NewConnectRecord and go to Step 12;
Step 11: (to try the record connected by the right pointer of record CurRecord)
         let i ← call the sub procedure2 self with parameters RootTree,
         Next(CurRecord) and CurRecord;
Step 12: stop (return i).

Sub Procedure3 (p, s)
Step 1:  if Next(p) = 0 (the right pointer of record p connects to nothing), let
             Next(p) ← s and stop;
Step 2:  call the sub procedure3 self with parameters Next(p) and s.
```

12.6 Conclusion

Many input data concerning resources should be specified for the simulation study.

The resources and costs required for the execution of an activity within a process definition are specified with the activity definition. Workflow participants of an activity are resolved at run-time to people or a Notes agent from an editor assigned to the activity. Unlike the Espresso workflow engine that allocates a work to all members of a team, the simulator allocates the work only to the workflow participants of the team.

The given execution time of an activity is used by the Espresso simulator to generate the time taken by the workflow participants for the executing 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 materials may postpone starting execution of an activity, and the routing-synchronous may delay routing or terminating a completed work.

The specified delay of an activity is the time period that must 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 upon the hourly costs defined in the PAVONE Organization Databases. Other costs are included in the fixed costs of 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 a PAVONE Organization Database or a Notes Organization Directory.

A work associated with a process definition will be allocated to the dummy participant of an editor, 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 a simulation run. The queue of a human workflow participant will be graphically simulated and therefore the data structure and algorithms of the queue was discussed in the chapter.

13 OVERALL SIMULATION MODEL

The Espresso simulation model (the Espresso simulator) implemented in PM allows process instances in accordance with process definitions to be graphically simulated or animated. The process instances of a process definition can be simulated in one or multiple Espresso application databases. Simulated resources are retrieved from PAVONE 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 a simulation run, dynamic state variables of the simulated Espresso WfMS can be graphically displayed on the process maps and in the resource window. After the simulation run, statistical simulation reports are created. They can be presented in chart or table forms and can be saved and/or printed. Simulation reports summarize the simulated system performance and can be utilized for analyzing bottlenecks and costs of the WfMS in order to improve the process definitions.

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 workflow participant to workflow participant, in accordance with the process definitions. The following three alternative simulation modes can be experimented in PM.

- Animate one process instance: one process instance will be created in an Espresso 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 system analyst should interact in decision-making;
- Simulate the process instances of one process definition: any number of process instances in accordance with a process definition can be simulated in one or multiple Espresso application databases. No user interaction is allowed during the simulation run;
- Simulate the process instances of all opened process definitions: process instances in accordance with different process definitions will be simulated in diverse Espresso application databases. No user interaction is allowed.

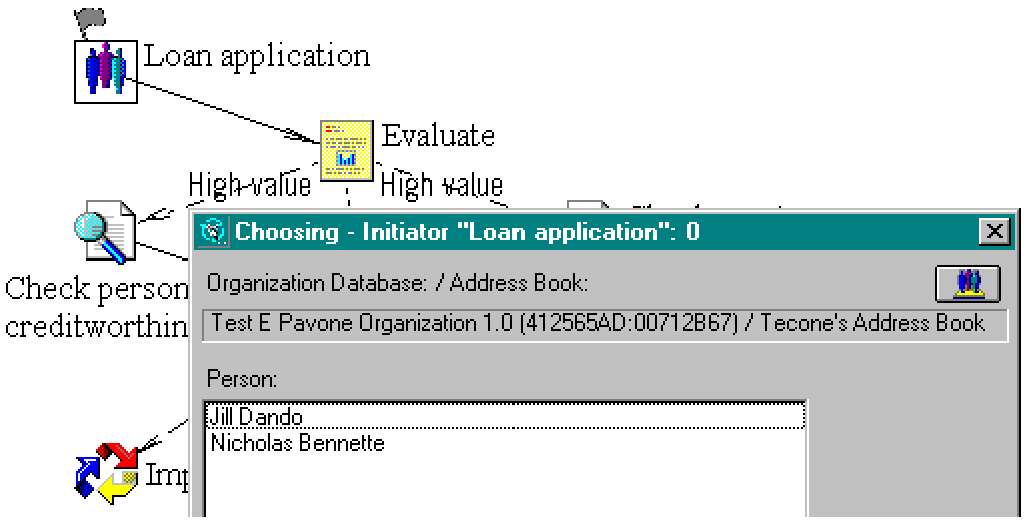
In all the experiment modes, routed work items associated with the process definitions can be animated and some state/statistic variables can be presented visually on the process maps and graphically in a resource window.

13.1.1 Animation

Animation is used to test a process definition step by step and to watch how a process definition will be activated in an application database of the Espresso WfMS. During animation, the system analyst should interact in the experiment to make diverse decisions.

- Process creation decision: determine a workflow participant (initiator) 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 must complete the activity (see Figure 13-2).
- Variate decision: choose the value of a variable defined in the formula of an "Condition" outgoing link of the activity (task) that is just completed (see Figure 13-3).
- Branching decision: choose "Multiple Choice" outgoing links and/or an "Exclusive Choice" outgoing link of an activity (task), over which the work at 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.

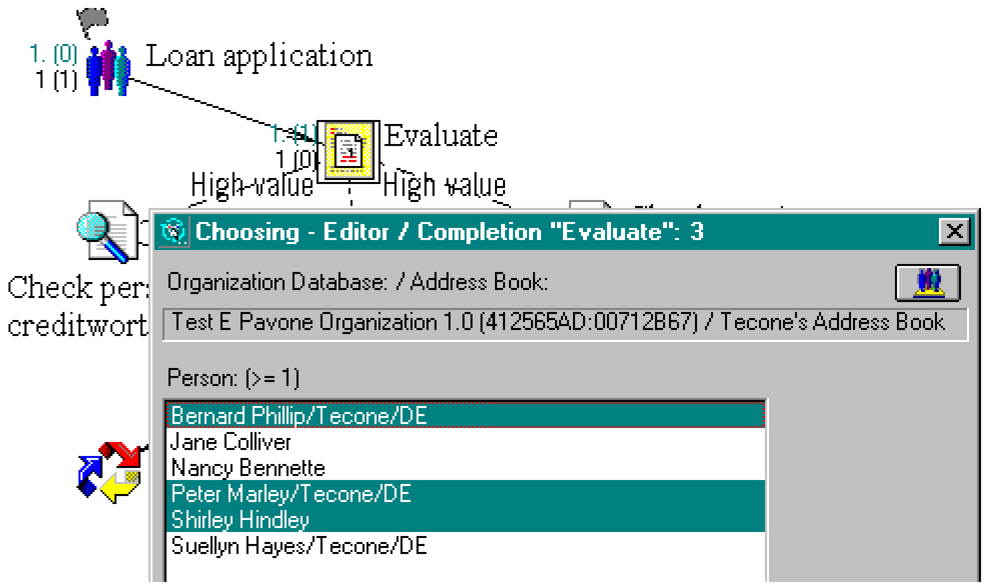
After animation, a process protocol from the activity where the process instance was created till the last completed activity will always be generated. The statistical simulation reports will also be created although just one process instance is simulated.


Loan application
Evaluate
High-value
High value
Check person creditworthin
Imp
Choosing - Initiator "Loan application": 0
Organization Database: / Address Book:
Person:

Test E Pavone Organization 1.0 (412565AD:00712B67) / Tecone's Address Book

Jill Dando
Nicholas Bennette

Figure 13-1. Choose Creator of a Process Instance



Choosing - Editor / Completion "Evaluate": 3

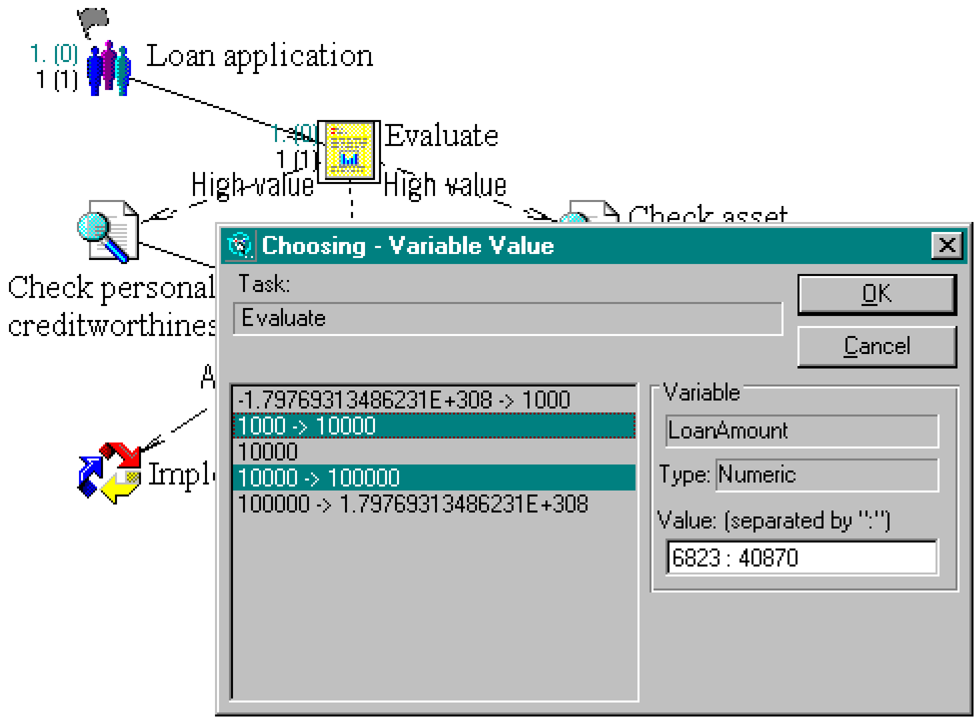
Organization Database: / Address Book:

Test E Pavone Organization 1.0 (412565AD:00712B67) / Tecone's Address Book

Person: (>= 1)

Bernard Phillip/Tecone/DE
Jane Colliver
Nancy Bennette
Peter Marley/Tecone/DE
Shirley Hindley
Suellyn Hayes/Tecone/DE

Figure 13-2. Choose Workflow Participants of an Activity



Choosing - Variable Value

Task:

Evaluate

-1.797693134862 31E+308 -> 1000
1000 -> 10000
10000
10000 -> 100000
100000 -> 1.79769313486231E+308

Variable: LoanAmount

Type: Numeric

Value: (separated by ":")

6823 : 40870

Figure 13-3. Choose Variate of a Variable

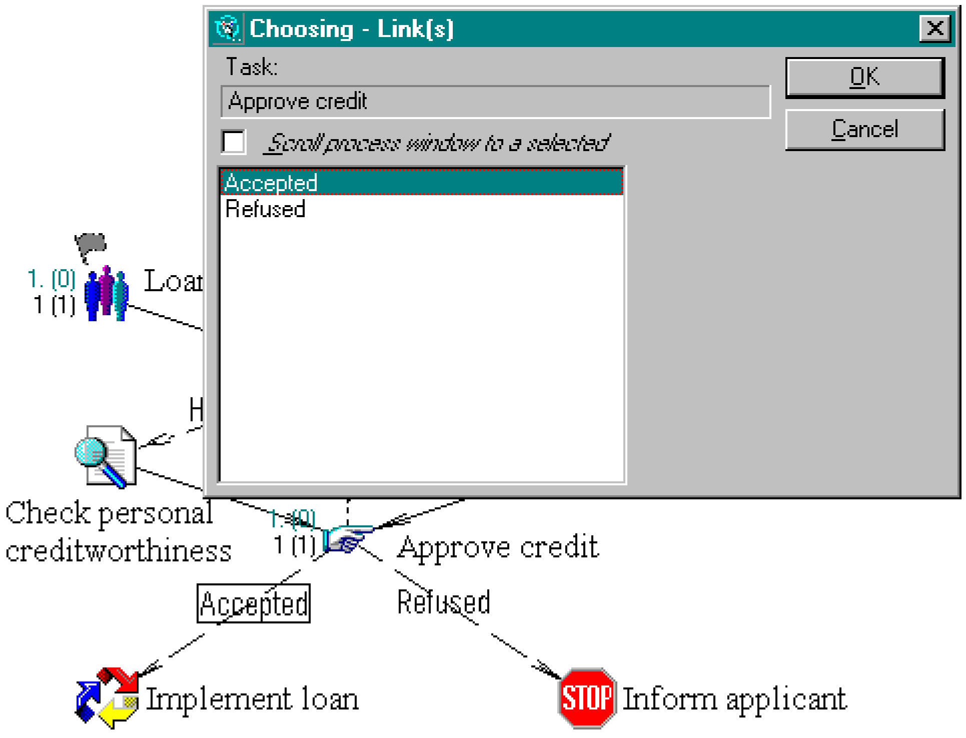


Figure 13-4. Choose Routing Links

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;
- to examine a join activity in order to analyze whether the duration of different execution threads of a process instance are balanced or not, or to detect the critical paths;
- to test whether a process definition will be activated as expected, especially where parallel work items are joined, how join priorities influence the release of deadlock situations, etc.;
- to experiment with a process definition in order to see what kind of decisions should be made at run-time by workflow participants, and how different decisions will influence the execution threads of a process instance in accordance with the process definition;
- to learn and evaluate the simulation results, since animation is the simplest simulation mode and thus the statistical results can be comprehended most easily; or
- to forecast how alternative operating policies will affect the business processes in accordance with the process definition.

13.1.2 Simulation

Process instances of 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 system analyst to observe how different resources, processes and activities will interact in a WfMS.

During a simulation run the system analyst cannot interact to make decisions as in animation. All these decisions are made stochastically by the simulator according to the corresponding distribution functions.

Simply to simulate the process instances of one process definition in one Espresso application database can help the system 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 simulator as the input data for a simulation study:

- specifications in activity definitions: 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) (see Section 12.1);
- specifications in link definitions: routing option and time (see Section 3.1);
- the organizational model: definitions of organizational roles and material resources, and hourly costs of human/material resources (see Section 1.2.3);
- process settings: life period, process intercreation time, routing probabilities of “Multiple Choice” links, routing distribution of “Exclusive Choice” links, and value distributions of variables defined in “Condition” links (see Section 11.1);
- resource settings: time distribution functions (for execution/delay of activities, routing of work items, and use of materials respectively), participating and simulating resources, work allocation distribution within a team, and public holidays (see Section 12.3);
- experimental settings: the unit of the clock, period of a simulation run, protocol-generating intervals, and the Espresso application databases as well as configured PAVONE Organization Databases and/or Notes Organization Directories for each analyzed process definition (see Section 13.2.1);

- simulation view settings: for displaying visually the simulated states of a WfMS, animating 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 in an Espresso Process Database. Organizational models are defined in PAVONE Organization Databases and/or Notes Organization Directories. Process settings, resource settings and experimental settings are stored in the Espresso Simulation Database. Simulation view settings are user preference settings for a simulation run. They can be modified during a simulation run and hence are not saved in the Espresso Simulation Database.

Apart from simulation view settings and some experimental settings, all others are input variables of the Espresso 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, or even to lead to a wrong decision.

13.2.1 Experimental Settings

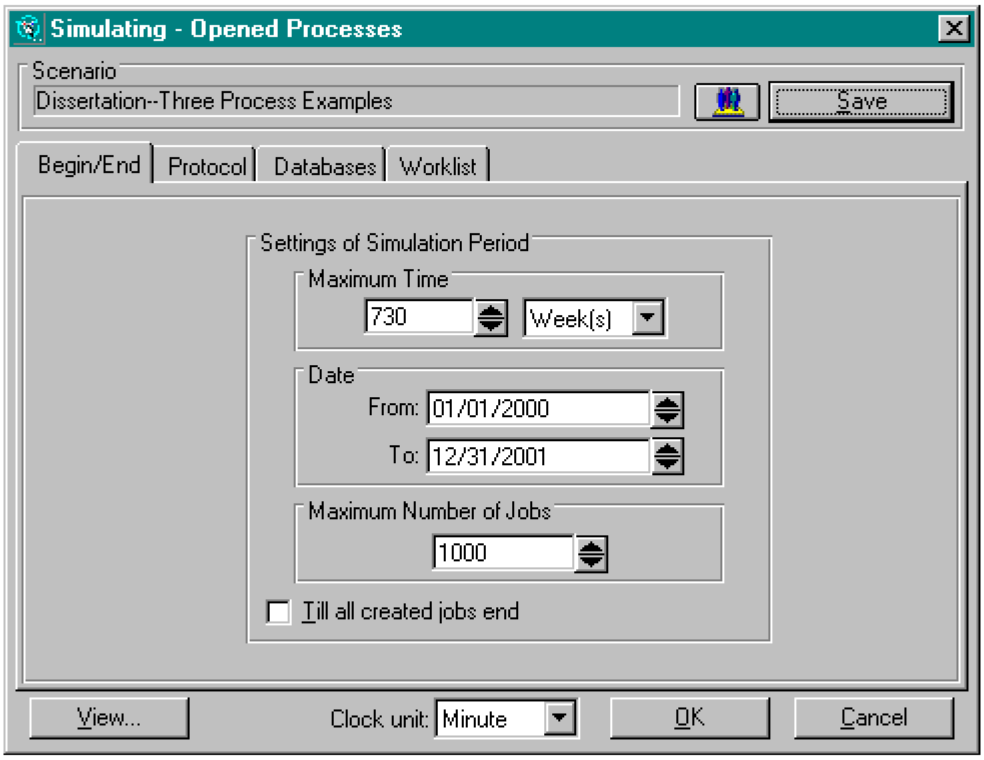
The experimental settings are prepared for a simulation run. They include the time unit of the clock, simulation beginning/ending conditions, interval to generate process protocols at a specified activity, databases configured to the analyzed process definitions, and queuing rules of simulated worklists of workflow participants.

The experimental settings can be saved as a *scenario* so that they can be used again for further simulation runs of the same scenario. Different simulation runs of the same scenario may generate different simulation results, if the simulated WfMS contains any random variable. In this case, multiple runs of a scenario are necessary for the evaluation of the simulation results.

13.2.1.1 Simulation Beginning/Ending Conditions

Simulation beginning/ending conditions determine the period of a simulation run. For a simulation run, the time period when a process instance can be created, or the maximum number of process instances (jobs) to be created, can be specified in the dialogue box 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.


Simulating - Opened Processes

Scenario: Dissertation--Three Process Examples

Settings of Simulation Period

Maximum Time: 730 Week(s)

Date From: 01/01/2000 To: 12/31/2001

Maximum Number of Jobs: 1000

☐ Till all created jobs end

Clock unit: Minute

Figure 13-5. Simulation Beginning/Ending Settings

If the simulation beginning date is specified, a simulation run is assumed to begin on the date corresponding to the clock value zero. Therefore, 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 common public holidays. The clock can be displayed in date form during the simulation run 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, so that a simulation run can automatically end. 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 in accordance with a process definition does not exceed the given maximum number of process instances (jobs).

That is, no more new process instances will be created in the simulated Espresso WfMS, if one of the stipulated conditions arises.

If the maximum simulation time or the simulation ending date is given, the system 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 terminated, and thus the simulation run can end.

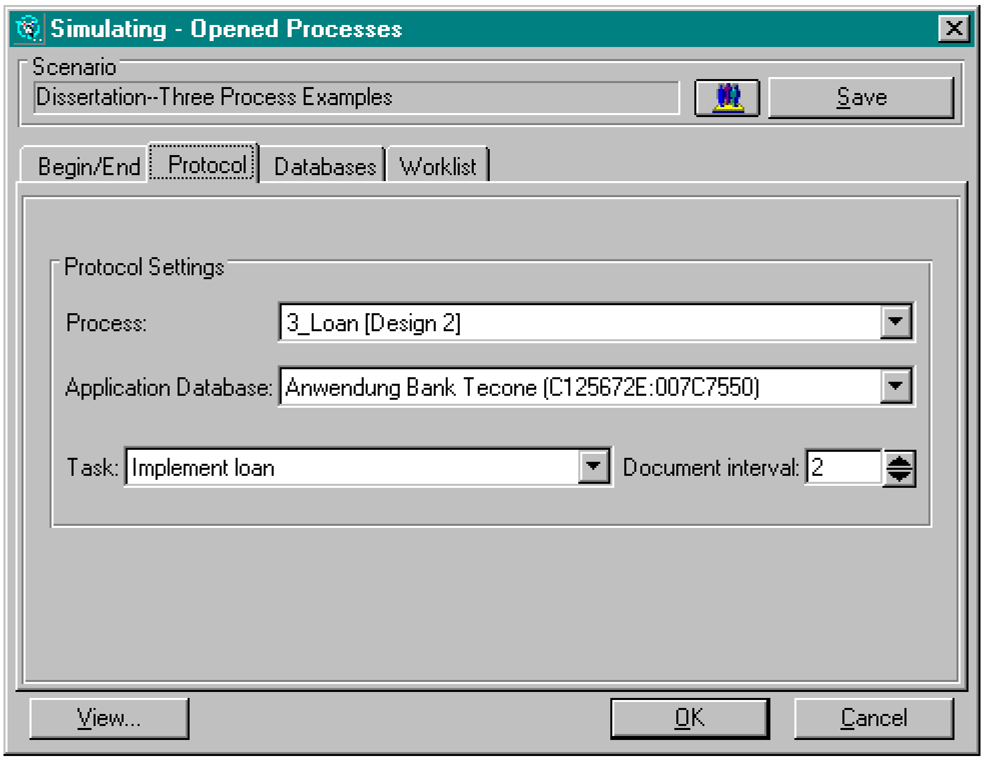
In addition to above specified conditions, the creations of process instances in accordance with a process definition are simulated within the defined life period of the process definition. For example, suppose that a process definition has life ending date 12/31/1999. According to the above experimental settings in Figure 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 as well as the simulation ending date works together with the simulation beginning date. Therefore, if an analyzed 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 analyzed 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

A process protocol (see Section 13.3.1) presents an execution thread of a process instance from the start activity up to the activity (task) specified in the process protocol settings as shown in Figure 13-6. The interval of work items at



Scenario: Dissertation--Three Process Examples

Protocol Settings

Process: 3_Loan [Design 2]

Application Database: Anwendung Bank Tecone (C125672E:007C7550)

Task: Implement loan   Document interval: 2

Figure 13-6. Process Protocol Settings

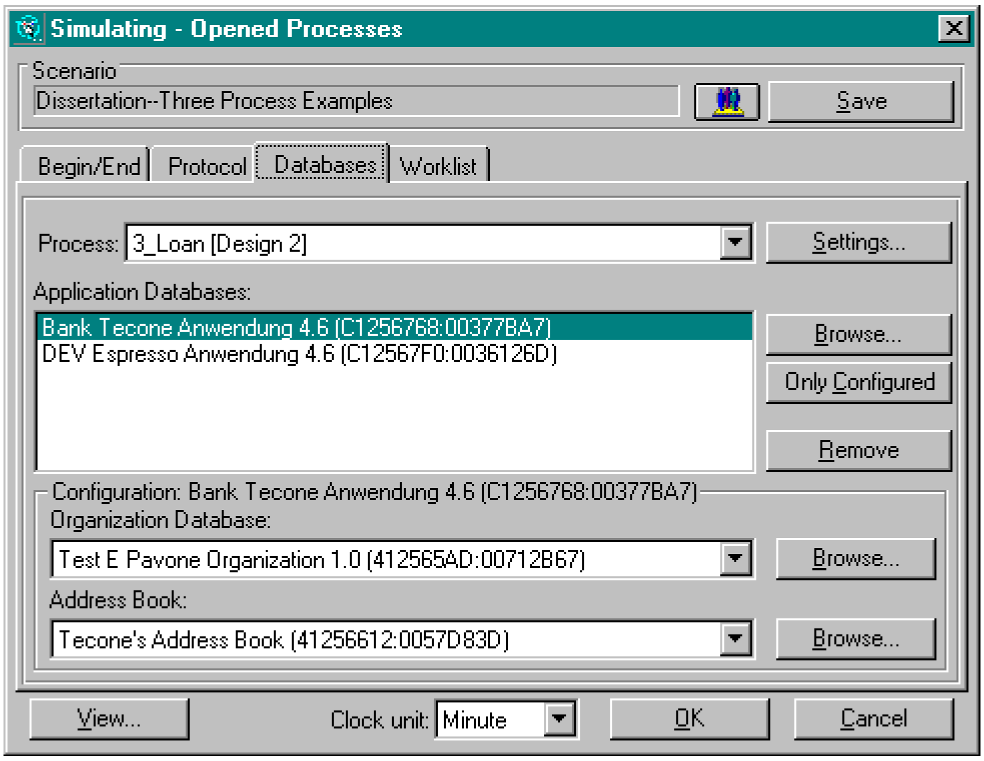
the activity is also specified here for generating the process protocols. The specification is related to an Espresso application database where the process instances of the process definition are simulated. In the example of Figure 13-6, a process protocol will be generated at activity "Implement loan", when every second work (represented by a document in the Espresso WfMS) 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 Espresso Simulation Database and can be displayed in table form during simulation or animation.

13.2.1.3 Database Settings

The Espresso application databases in which process instances of a process definition will be simulated are specified here. The PAVONE Organization Database and/or Notes Organization Directory configured to the Espresso application database are utilized for retrieving simulated human and material resources. The application databases as well as the configurations are specified in the dialogue box shown in Figure 13-7.



Scenario: Dissertation--Three Process Examples

Process: 3_Loan [Design 2]

Application Databases:
Bank Tecone Anwendung 4.6 (C1256768:00377BA7)
DEV Espresso Anwendung 4.6 (C12567F0:0036126D)

Configuration: Bank Tecone Anwendung 4.6 (C1256768:00377BA7)
Organization Database: Test E Pavone Organization 1.0 (412565AD:00712B67)
Address Book: Tecone's Address Book (41256612:0057D83D)

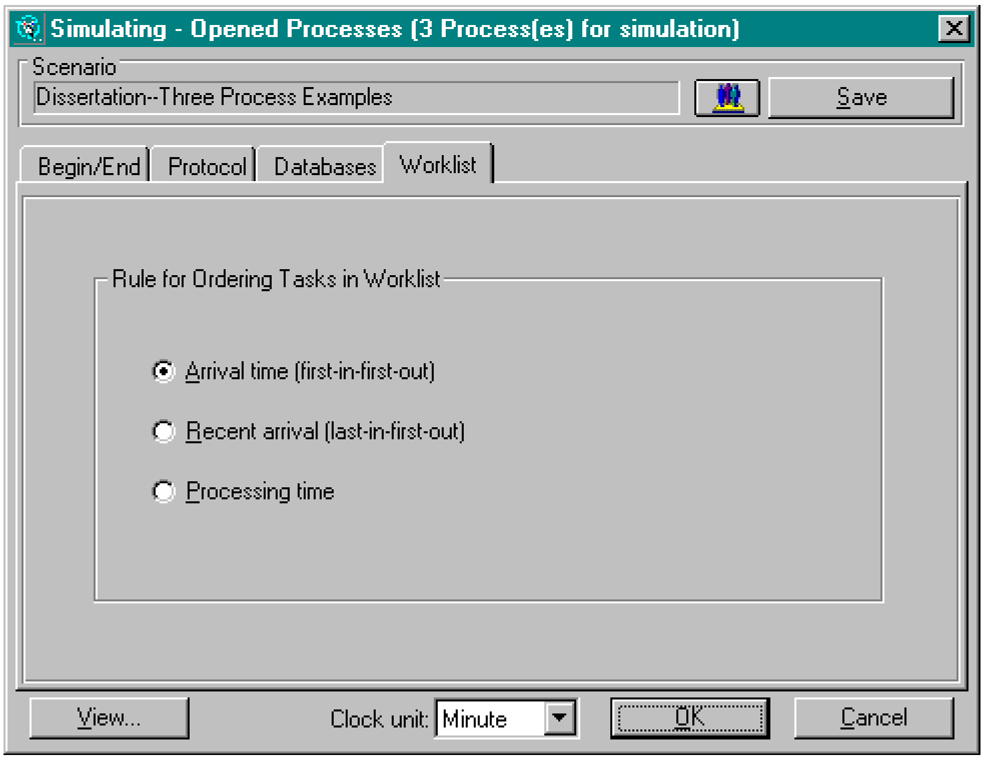
Clock unit: Minute

Figure 13-7. Database Configurations for Simulation

Process instances of a process definition can be simultaneously simulated in several Espresso application databases. In this example, process instances of process definition "Loan" will be simulated concurrently under two Espresso application databases with the same or different configurations.

13.2.1.4 Queuing Rule for Worklist

The principle of sorting the worklist of a workflow participant in an Espresso 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 or the last entry in the worklist. Therefore, different sorting principles can result in different system performances. Settings of queuing rule for worklists as shown in Figure 13-8 allow the system analyst to decide which


Simulating - Opened Processes (3 Process(es) for simulation)

Scenario: Dissertation--Three Process Examples

Rule for Ordering Tasks in Worklist: [✓ Arrival time (first-in-first-out)] Recent arrival (last-in-first-out) Processing time

Clock unit: Minute

Figure 13-8. Worklist Sorting Settings

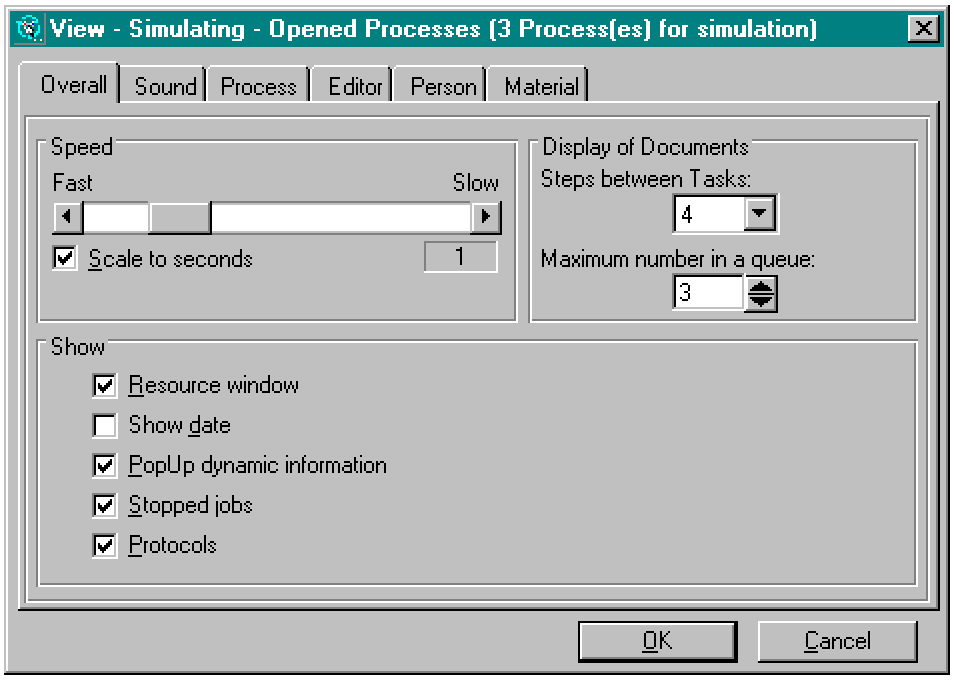
of the following principles is the best for sorting the worklists in an Espresso application database:

- in the order of arrival time (i.e. first-in-first-out),
- in the order of recent arrival (i.e. 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 that can be modified before and during a simulation run. Simulation view settings specified in the dialogue


View - Simulating - Opened Processes (3 Process(es) for simulation)

Overall | Sound | Process | Editor | Person | Material

Speed: Fast – Slow
[✓] Scale to seconds: 1

Display of Documents
Steps between Tasks: 4
Maximum number in a queue: 3

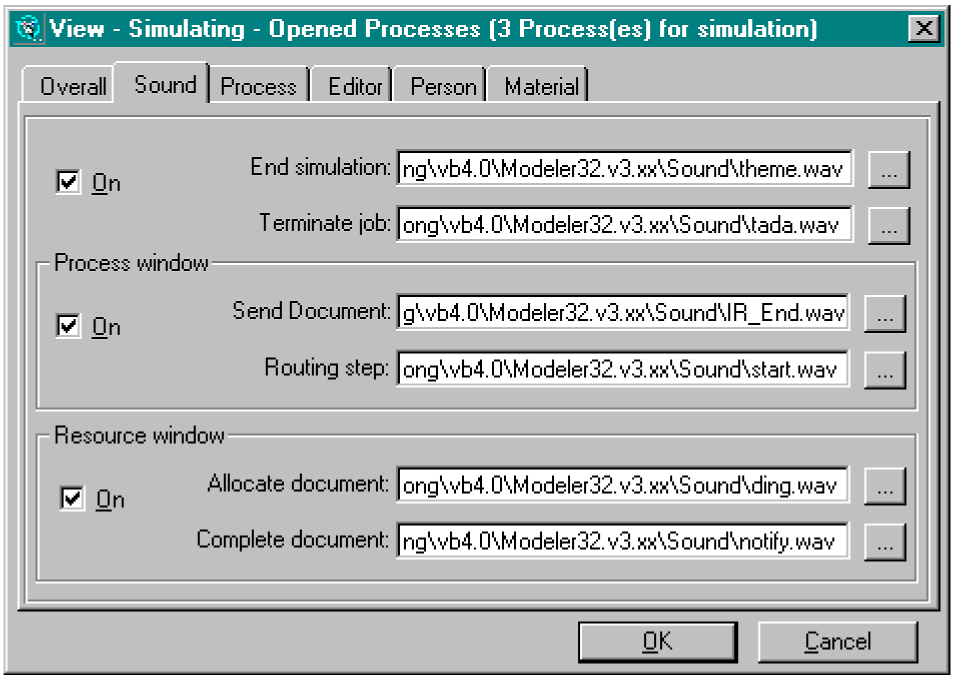
Show
[✓] Resource window
[ ] Show date
[✓] PopUp dynamic information
[✓] Stopped jobs
[✓] Protocols

OK | Cancel

Figure 13-9. Simulation View Overall Settings

box shown in Figure 13-9 include:

- 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 dynamically displayed data on process map windows and the resource window,
- whether to prompt messages about a stopped work associated with a process instance (job),
- whether to prompt generated process protocols,
- the sound for different events (see Figure 13-10),
- the displaying process map windows (see Figure 13-11), and
- the editors, human and material resources depicted in the resource window (similar to the settings shown in Figure 13-11).


View - Simulating - Opened Processes (3 Process(es) for simulation)

☑ On

End simulation: ng\vb4.0\Modeler32.v3.xx\Sound\theme.wav

Terminate job: ong\vb4.0\Modeler32.v3.xx\Sound\tada.wav

Process window

☑ On

Send Document: g\vb4.0\Modeler32.v3.xx\Sound\IR_End.wav

Routing step: ong\vb4.0\Modeler32.v3.xx\Sound\start.wav

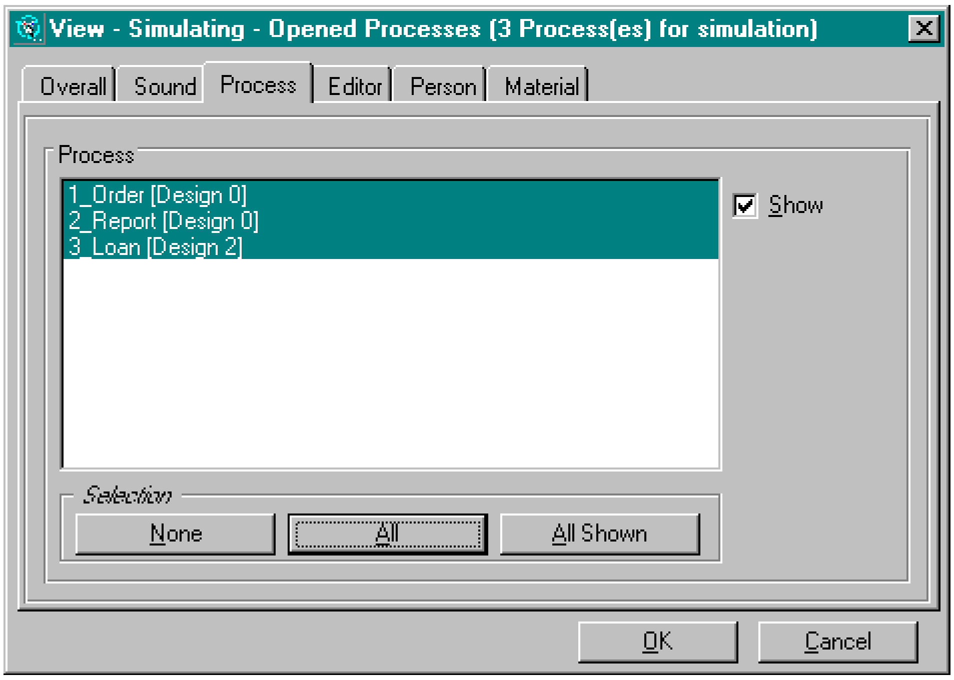
Resource window

☑ On

Allocate document: ong\vb4.0\Modeler32.v3.xx\Sound\ding.wav

Complete document: ng\vb4.0\Modeler32.v3.xx\Sound\notify.wav

Figure 13-10. Simulation Sound Settings


View - Simulating - Opened Processes (3 Process(es) for simulation)

Process

1_Order [Design 0]
2_Report [Design 0]
3_Loan [Design 2]

☑ Show

Selection

Figure 13-11. Simulation View Process Map Settings

13.3 Simulation Reports

The simulated process instances represent the business processes running in an organization that offers services and/or products. For such a WfMS, 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 the 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 a simulation run presents how an execution thread of a process instance is performed from the start activity where the process instance was created till the specified activity.

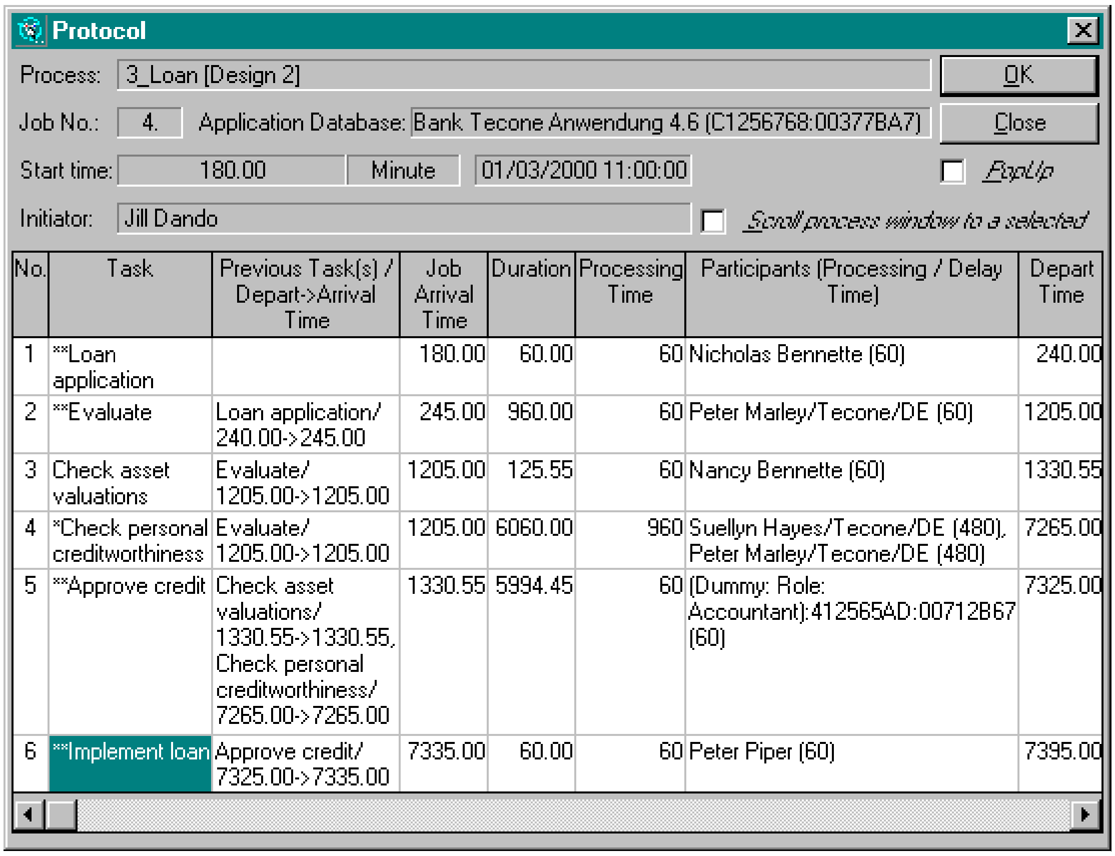
The statistical reports of a simulation run can be saved in the Espresso Simulation Database and can be open again. The generated process protocols will be saved with them. Before the reports are saved, the corresponding experiment settings (i.e. scenario) must be saved too.

The different reports shown later in the figures of this Section were the results of one simulation run. So they will be used together to analyze the bottlenecks of the simulated WfMS.

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 activity execution thread of a process instance (job). It presents when the process instance was 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 (task), the following data are contained 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 arrived from the predecessors;
- duration of the work associated with the activity;
- 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 was routed out from the activity (depart time).


Protocol

Process: 3_Loan [Design 2]

Job No.: 4. Application Database: Bank Tecone Anwendung 4.6 (C1256768:00377BA7)

Start time: 180.00 Minute 01/03/2000 11:00:00

Initiator: Jill Dando

No.	Task	Previous Task(s) / Depart->Arrival Time	Job Arrival Time	Duration	Processing Time	Participants (Processing / Delay Time)	Depart Time
1	**Loan application		180.00	60.00	60	Nicholas Bennette (60)	240.00
2	**Evaluate	Loan application/ 240.00->245.00	245.00	960.00	60	Peter Marley/Tecone/DE (60)	1205.00
3	Check asset valuations	Evaluate/ 1205.00->1205.00	1205.00	125.55	60	Nancy Bennette (60)	1330.55
4	*Check personal creditworthiness	Evaluate/ 1205.00->1205.00	1205.00	6060.00	960	Suellyn Hayes/Tecone/DE (480), Peter Marley/Tecone/DE (480)	7265.00
5	**Approve credit	Check asset valuations/ 1330.55->1330.55, Check personal creditworthiness/ 7265.00->7265.00	1330.55	5994.45	60	(Dummy: Role: Accountant):412565AD:00712B67 (60)	7325.00
6	**Implement loan	Approve credit/ 7325.00->7335.00	7335.00	60.00	60	Peter Piper (60)	7395.00

Figure 13-12. Process Protocol

The parallel execution threads within a process instance are included in the process protocol too. The data in the column "Previous Task(s)/Depart→Arrival Time" is a set of previous activities, with corresponding times departing from the previous and arriving at 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" (item 5 in the table).

When the process protocol is prompted, relevant activities and link on the process map will be square-marked (see Figure 13-13). Marked links present

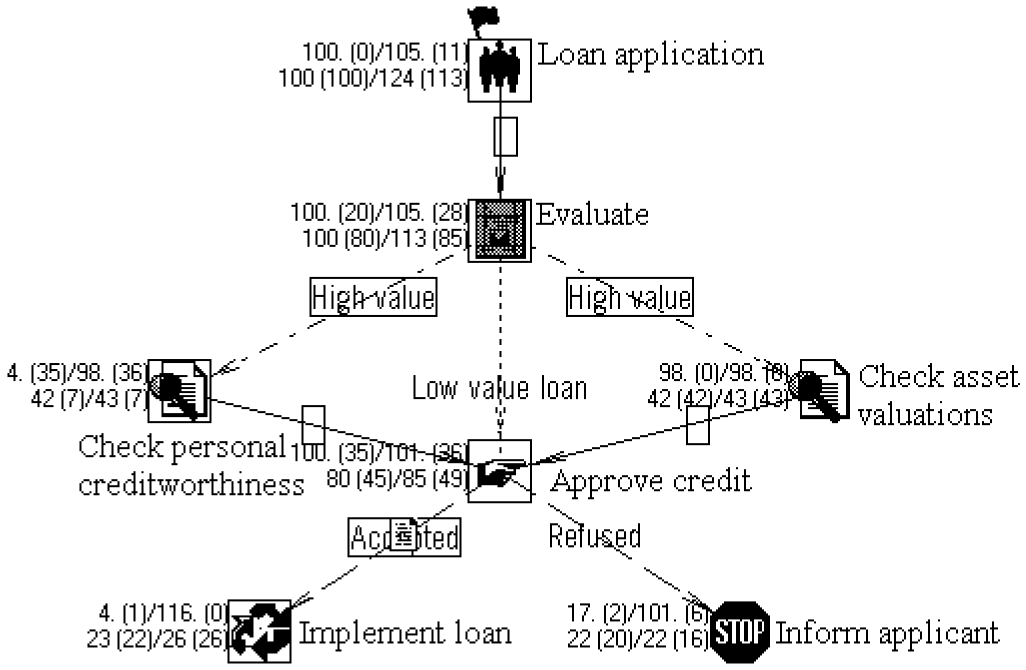


Figure 13-13. Process Definition and Process Protocol

graphically how the work associated with the process instance flow from activity to activity, in accordance with the process definition. Marked activities combine the critical path of the activity execution thread of the process instance.

In this example, the process instances of the process definition are simulated simultaneously in two Espresso application databases, and the data beside each activity in the map corresponds to the two databases respectively (for the descriptions of the data see Figure 11-5 in Section 11.2). From the process protocol in 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". Thus, it can be concluded that the first part of data beside each activity corresponds to this application database. (This can also be known from the pop up description of the dynamic data.)

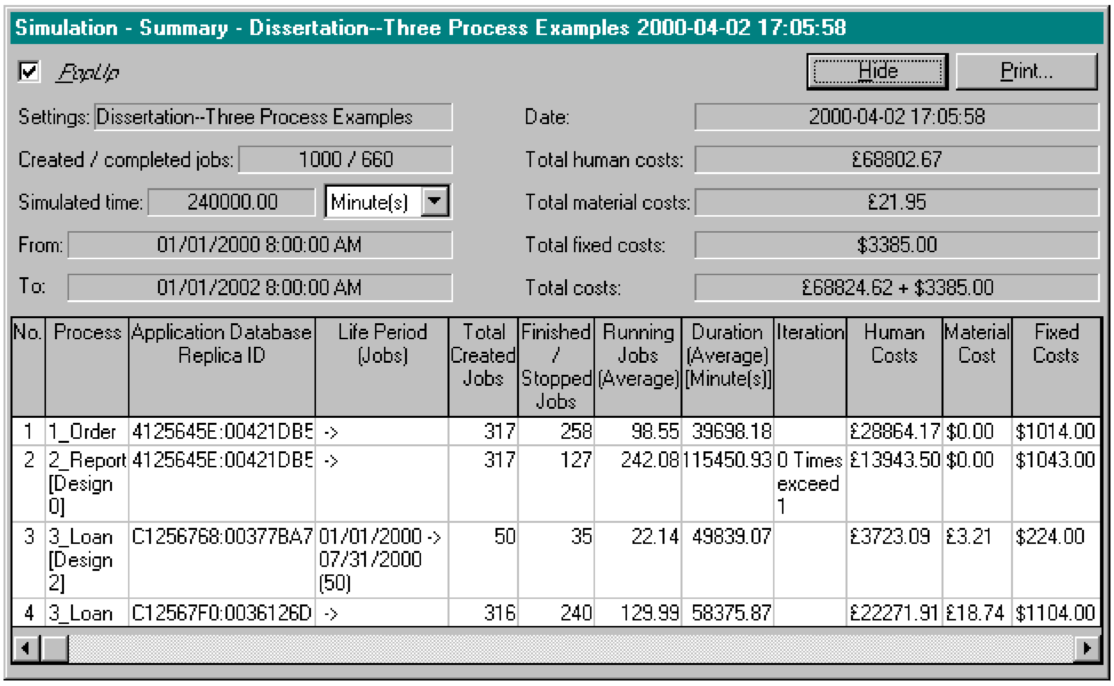
Duration of a work associated with an activity is the time period between the arrival time of the (first parallel) work and the depart time of the (joined) work from the activity. During this time period, multiple workflow participants of a team may execute the activity simultaneously. Duration includes also specified delay time of the activity and times waiting for joining, for resource use, and 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" (time of the last arrived subtracts that of 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 cannot execute the activity soon after the work is allocated in their worklist—they may have too many work items to do, or there is a shortage of the synchronous materials.

Long duration of a process instance causes 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 overall 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 hence should be reduced. To let the activity on the critical path depart the activity earlier can reduce the overall duration of the process instance. On the other hand, prolonging the duration of one activity involved in the critical path will lengthen the overall duration of a process instance.

Process protocols are saved in the Espresso Simulation Database with other simulation results. They can be displayed during a simulation run.

13.3.2 Simulation Summary Report

The simulation summary report as shown in Figure 13-14 includes general


Simulation - Summary - Dissertation--Three Process Examples 2000-04-02 17:05:58

Settings:	Dissertation--Three Process Examples	Date:	2000-04-02 17:05:58
Created / completed jobs:	1000 / 660	Total human costs:	£68802.67
Simulated time:	240000.00 Minute(s)	Total material costs:	£21.95
From:	01/01/2000 8:00:00 AM	Total fixed costs:	$3385.00
To:	01/01/2002 8:00:00 AM	Total costs:	£68824.62 + $3385.00

No.	Process	Application Database Replica ID	Life Period (Jobs)	Total Created Jobs	Finished / Stopped Jobs	Running Jobs (Average)	Duration (Average) [Minute(s)]	Iteration	Human Costs	Material Cost	Fixed Costs
1	1_Order	4125645E:00421DB5	->	317	258	98.55	39698.18		£28864.17	$0.00	$1014.00
2	2_Report [Design 0]	4125645E:00421DB5	->	317	127	242.08	115450.93	0 Times exceed 1	£13943.50	$0.00	$1043.00
3	3_Loan [Design 2]	C1256768:00377BA7	01/01/2000 -> 07/31/2000 (50)	50	35	22.14	49839.07		£3723.09	£3.21	$224.00
4	3_Loan	C12567F0:0036126D	->	316	240	129.99	58375.87		£22271.91	£18.74	$1104.00

Figure 13-14. Simulation Summary Report

results of a simulation run. They are:

- name of the relevant simulation scenario (settings),
- generating time of the simulation reports,
- total created and completed process instances (jobs),

- period of the simulation run,
- start and end date of the simulation run,
- total costs of the simulated WfMS, and
- the list of the analyzed process definitions under different simulated application databases.

The example in Figure 13-14 presents the summary report after simulating process instances of the three process definitions in Figure 1-3, Figure 1-4 and Figure 1-5 in Section 1.3). Process instances of process definition “Loan” were simulated in two Espresso application databases with the different Replica IDs.

In the summary report, system performance data associated with each process definition are summarized in the table from the activity report (see Section 13.3.3) and from the resource report (see Section 13.3.4). Statistical data for a process definition includes:

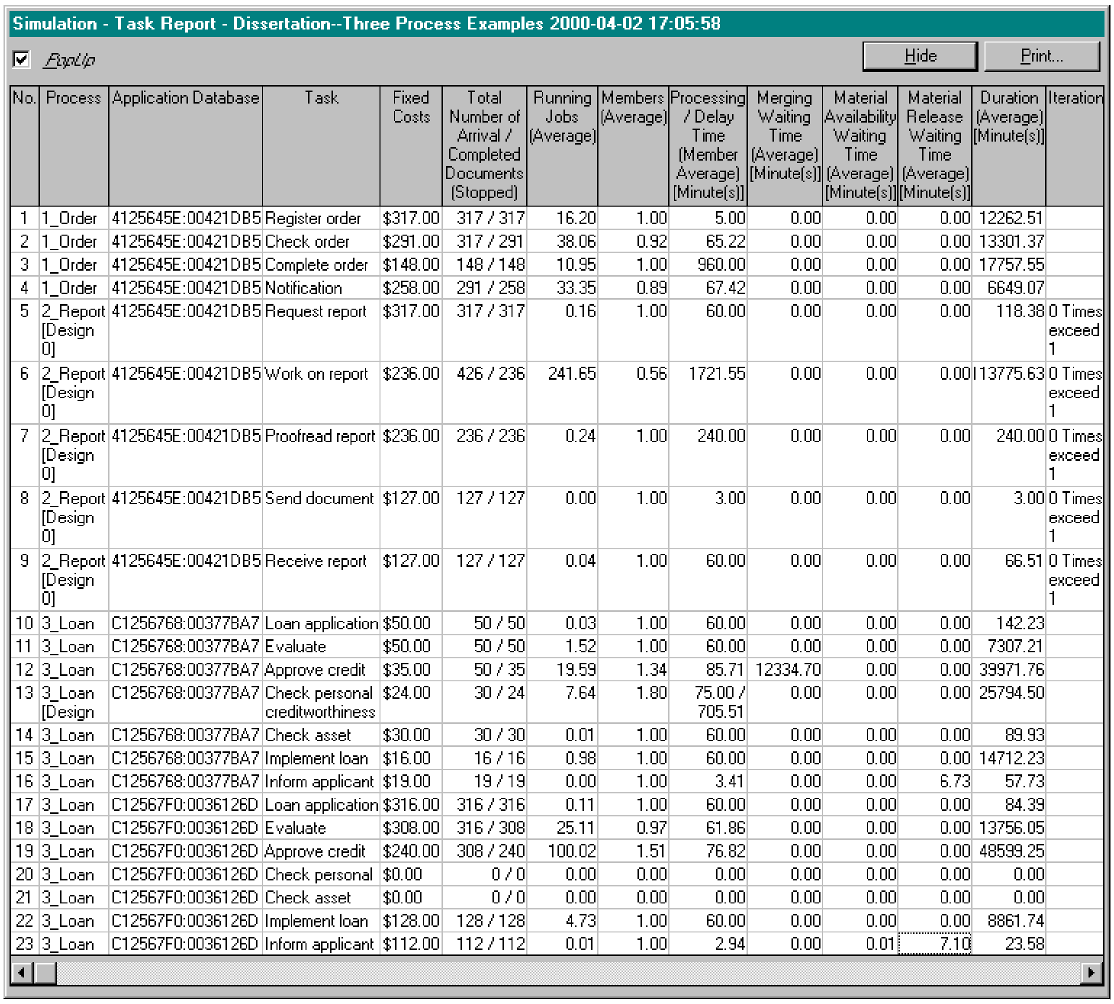
- the application database in which the process instances of the process definition were simulated,
- life period of the process definition,
- total simulated process instances (jobs),
- number of terminated (finished and stopped respectively) process instances,
- average number of running process instances in the system,
- average duration of a process instance,
- total costs of human and material resources,
- total fixed costs, etc.

The report includes also some escalation data, in order to raise necessary alarms for the simulated Espresso WfMS. Escalation values relevant to a process definition are accumulated when a process instance is terminated. In this example, the escalation condition is: iteration of an activity within the process instance is more than one. Of all 127 completed process instances in accordance with process definition “Report” (item 2 in the table), no process instance met this escalation condition—some of the 90 (= 317 − 127) not terminated process instances might meet the escalation condition, since the iteration prolongs the duration of a process instance. To know where the iterations have happened, the activity report can be further analyzed.

13.3.3 Activity Report

The activity report shown in Figure 13-15 collects statistical data of every activity (task) within the analyzed process definitions. For each activity there are:

- fixed costs;
- total number of created and completed activity instances (represented by document in the Espresso WfMS);
- average number of running work items (jobs) associated with the activity, in the simulated system;
- average number of team members having executed the activity;
- average execution and delay time by a workflow participant;
- average join-waiting time;
- average material availability-waiting time;
- average material release-waiting time;
- average duration;
- number of process instances containing execution iterations of the activity exceeding the maximum number of iterations (here 1) specified to the process definition; etc.


Simulation - Task Report - Dissertation--Three Process Examples 2000-04-02 17:05:58
PopUp
Hide
Print...

No.	Process	Application Database	Task	Fixed Costs	Total Number of Arrival / Completed Documents (Stopped)	Running Jobs (Average)	Members (Average)	Processing / Delay Time (Member Average) [Minute(s)]	Merging Waiting Time (Average) [Minute(s)]	Material Availability Waiting Time (Average) [Minute(s)]	Material Release Waiting Time (Average) [Minute(s)]	Duration (Average) [Minute(s)]	Iteration
1	1_Order	4125645E:00421DB5	Register order	$317.00	317 / 317	16.20	1.00	5.00	0.00	0.00	0.00	12262.51	
2	1_Order	4125645E:00421DB5	Check order	$291.00	317 / 291	38.06	0.92	65.22	0.00	0.00	0.00	13301.37	
3	1_Order	4125645E:00421DB5	Complete order	$148.00	148 / 148	10.95	1.00	960.00	0.00	0.00	0.00	17757.55	
4	1_Order	4125645E:00421DB5	Notification	$258.00	291 / 258	33.35	0.89	67.42	0.00	0.00	0.00	6649.07	
5	2_Report [Design 0]	4125645E:00421DB5	Request report	$317.00	317 / 317	0.16	1.00	60.00	0.00	0.00	0.00	118.38	0 Times exceed 1
6	2_Report [Design 0]	4125645E:00421DB5	Work on report	$236.00	426 / 236	241.65	0.56	1721.55	0.00	0.00	0.00	113775.63	0 Times exceed 1
7	2_Report [Design 0]	4125645E:00421DB5	Proofread report	$236.00	236 / 236	0.24	1.00	240.00	0.00	0.00	0.00	240.00	0 Times exceed 1
8	2_Report [Design 0]	4125645E:00421DB5	Send document	$127.00	127 / 127	0.00	1.00	3.00	0.00	0.00	0.00	3.00	0 Times exceed 1
9	2_Report [Design 0]	4125645E:00421DB5	Receive report	$127.00	127 / 127	0.04	1.00	60.00	0.00	0.00	0.00	66.51	0 Times exceed 1
10	3_Loan	C1256768:00377BA7	Loan application	$50.00	50 / 50	0.03	1.00	60.00	0.00	0.00	0.00	142.23	
11	3_Loan	C1256768:00377BA7	Evaluate	$50.00	50 / 50	1.52	1.00	60.00	0.00	0.00	0.00	7307.21	
12	3_Loan	C1256768:00377BA7	Approve credit	$35.00	50 / 35	19.59	1.34	85.71	12334.70	0.00	0.00	39971.76	
13	3_Loan [Design	C1256768:00377BA7	Check personal creditworthiness	$24.00	30 / 24	7.64	1.80	75.00 / 705.51	0.00	0.00	0.00	25794.50	
14	3_Loan	C1256768:00377BA7	Check asset	$30.00	30 / 30	0.01	1.00	60.00	0.00	0.00	0.00	89.93	
15	3_Loan	C1256768:00377BA7	Implement loan	$16.00	16 / 16	0.98	1.00	60.00	0.00	0.00	0.00	14712.23	
16	3_Loan	C1256768:00377BA7	Inform applicant	$19.00	19 / 19	0.00	1.00	3.41	0.00	0.00	6.73	57.73	
17	3_Loan	C12567F0:0036126D	Loan application	$316.00	316 / 316	0.11	1.00	60.00	0.00	0.00	0.00	84.39	
18	3_Loan	C12567F0:0036126D	Evaluate	$308.00	316 / 308	25.11	0.97	61.86	0.00	0.00	0.00	13756.05	
19	3_Loan	C12567F0:0036126D	Approve credit	$240.00	308 / 240	100.02	1.51	76.82	0.00	0.00	0.00	48599.25	
20	3_Loan	C12567F0:0036126D	Check personal	$0.00	0 / 0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
21	3_Loan	C12567F0:0036126D	Check asset	$0.00	0 / 0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
22	3_Loan	C12567F0:0036126D	Implement loan	$128.00	128 / 128	4.73	1.00	60.00	0.00	0.00	0.00	8861.74	
23	3_Loan	C12567F0:0036126D	Inform applicant	$112.00	112 / 112	0.01	1.00	2.94	0.00	0.01	7.10	23.58	

Figure 13-15. Activity Report

Fixed costs are directly proportional to the number of executed activity instances. If execution iteration of an activity within a process instance 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 arises when a workflow participant has completed an activity but cannot route the work at the activity further, because a routing-synchronous material has not been released from the current execution of that activity.

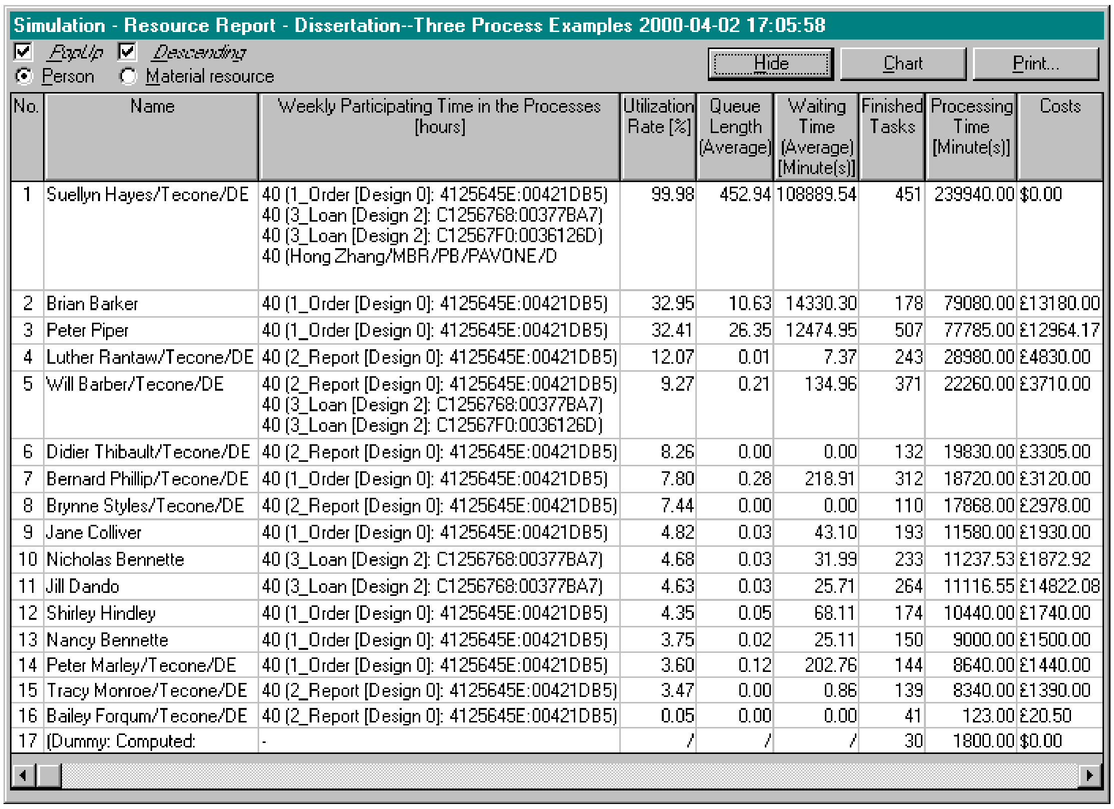
Average duration of an activity is affected by the execution time, delay time, join-waiting time, material availability-waiting time, and material release-waiting time. From this report, the reasons of the long duration of an activity can be analyzed.

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 the average execution and delay time may be larger than the average duration, because multiple workflow participants of a team can execute the activity simultaneously.
2. A large amount of time waiting for joining. For example, for activity "Approve credit" of process definition "Loan" simulated in the Espresso application database with Replica ID "C1256768:00377BA7" (item 12 in the table), the average execution time is 85.71 minutes, join-waiting time 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, via analyzing the process protocols;
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 of an activity are not balanced, or there is a shortage of material resources. For activity "Inform applicant" (item 23 in the table), for example, the average time waiting for material release is 7.10 minutes. This contributes to the average duration of 23.58 minutes;
5. Shortage of workflow participants for executing the activity, if none of the above outlined factors 1 through 4 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 in the activity report too. If there is execution iteration of an activity, 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 shown in Figure 13-16 indicates performance data of each simulated human and material resource, retrieved from the PAVONE


Simulation - Resource Report - Dissertation--Three Process Examples 2000-04-02 17:05:58
☑ PopUp ☑ Descending
● Person ○ Material resource
Hide
Chart
Print...

No.	Name	Weekly Participating Time in the Processes [hours]	Utilization Rate [%]	Queue Length (Average)	Waiting Time (Average) [Minute(s)]	Finished Tasks	Processing Time [Minute(s)]	Costs
1	Suellyn Hayes/Tecone/DE	40 (1_Order [Design 0]: 4125645E:00421DB5) 40 (3_Loan [Design 2]: C1256768:00377BA7) 40 (3_Loan [Design 2]: C12567F0:0036126D) 40 (Hong Zhang/MBR/PB/PAVONE/D	99.98	452.94	108889.54	451	239940.00	$0.00
2	Brian Barker	40 (1_Order [Design 0]: 4125645E:00421DB5)	32.95	10.63	14330.30	178	79080.00	£13180.00
3	Peter Piper	40 (1_Order [Design 0]: 4125645E:00421DB5)	32.41	26.35	12474.95	507	77785.00	£12964.17
4	Luther Rantaw/Tecone/DE	40 (2_Report [Design 0]: 4125645E:00421DB5)	12.07	0.01	7.37	243	28980.00	£4830.00
5	Will Barber/Tecone/DE	40 (2_Report [Design 0]: 4125645E:00421DB5) 40 (3_Loan [Design 2]: C1256768:00377BA7) 40 (3_Loan [Design 2]: C12567F0:0036126D)	9.27	0.21	134.96	371	22260.00	£3710.00
6	Didier Thibault/Tecone/DE	40 (2_Report [Design 0]: 4125645E:00421DB5)	8.26	0.00	0.00	132	19830.00	£3305.00
7	Bernard Phillip/Tecone/DE	40 (1_Order [Design 0]: 4125645E:00421DB5)	7.80	0.28	218.91	312	18720.00	£3120.00
8	Brynne Styles/Tecone/DE	40 (2_Report [Design 0]: 4125645E:00421DB5)	7.44	0.00	0.00	110	17868.00	£2978.00
9	Jane Colliver	40 (1_Order [Design 0]: 4125645E:00421DB5)	4.82	0.03	43.10	193	11580.00	£1930.00
10	Nicholas Bennette	40 (3_Loan [Design 2]: C1256768:00377BA7)	4.68	0.03	31.99	233	11237.53	£1872.92
11	Jill Dando	40 (3_Loan [Design 2]: C1256768:00377BA7)	4.63	0.03	25.71	264	11116.55	£14822.08
12	Shirley Hindley	40 (1_Order [Design 0]: 4125645E:00421DB5)	4.35	0.05	68.11	174	10440.00	£1740.00
13	Nancy Bennette	40 (1_Order [Design 0]: 4125645E:00421DB5)	3.75	0.02	25.11	150	9000.00	£1500.00
14	Peter Marley/Tecone/DE	40 (1_Order [Design 0]: 4125645E:00421DB5)	3.60	0.12	202.76	144	8640.00	£1440.00
15	Tracy Monroe/Tecone/DE	40 (2_Report [Design 0]: 4125645E:00421DB5)	3.47	0.00	0.86	139	8340.00	£1390.00
16	Bailey Forqum/Tecone/DE	40 (2_Report [Design 0]: 4125645E:00421DB5)	0.05	0.00	0.00	41	123.00	£20.50
17	(Dummy: Computed:	-	/	/	/	30	1800.00	$0.00

Figure 13-16. Resource Report

Organization Database and/or Notes Organization Directory configured to the Espresso application database in which the process instances of a process definition were simulated.

Statistical data for each simulated resource includes:

- utilization rate during the period of the simulation run,
- average queue length 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 all different activities (tasks),
- total execution time, and
- costs for the utilization.

Resource cost is proportioned to total executed time since hourly costs of a resource can be retrieved from the configured PAVONE Organization Database.

If a resource is demanded by too many work items associated with various activities, its queue length and waiting time will rise, and performance of the system will become worse. However a low utilization rate of a resource means waste of resource costs. Usually a resource with a long waiting queue has high utilization rate. If not, balance problems may exist between human and material resources, both are required simultaneously for activity execution.

The table of the resource report 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 better comprehension and easy comparison.

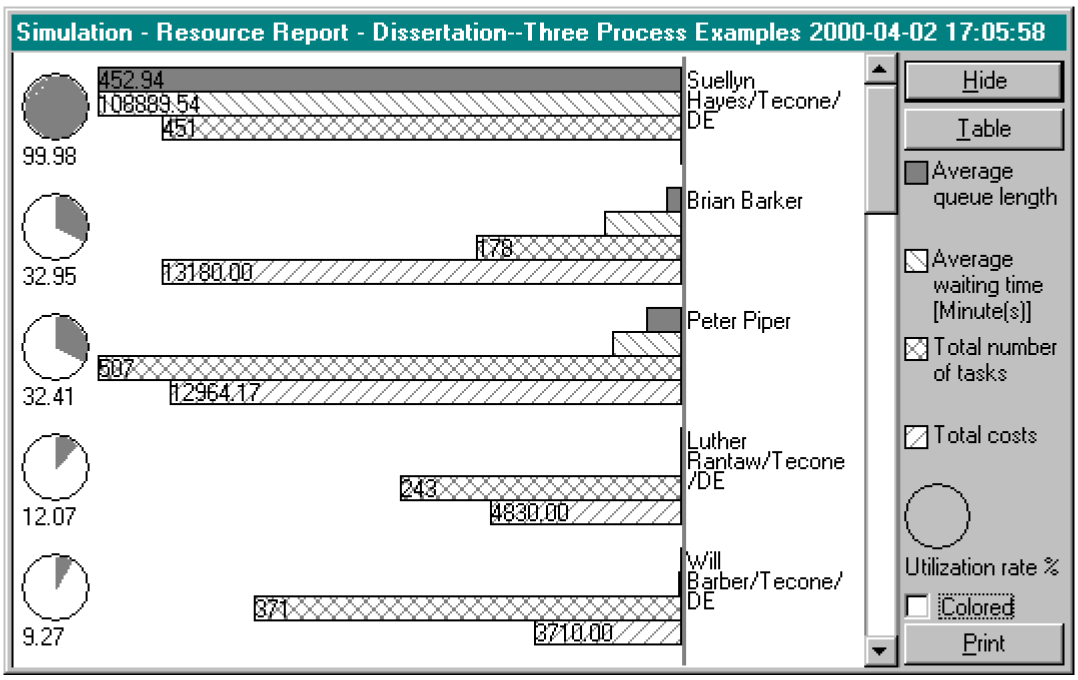


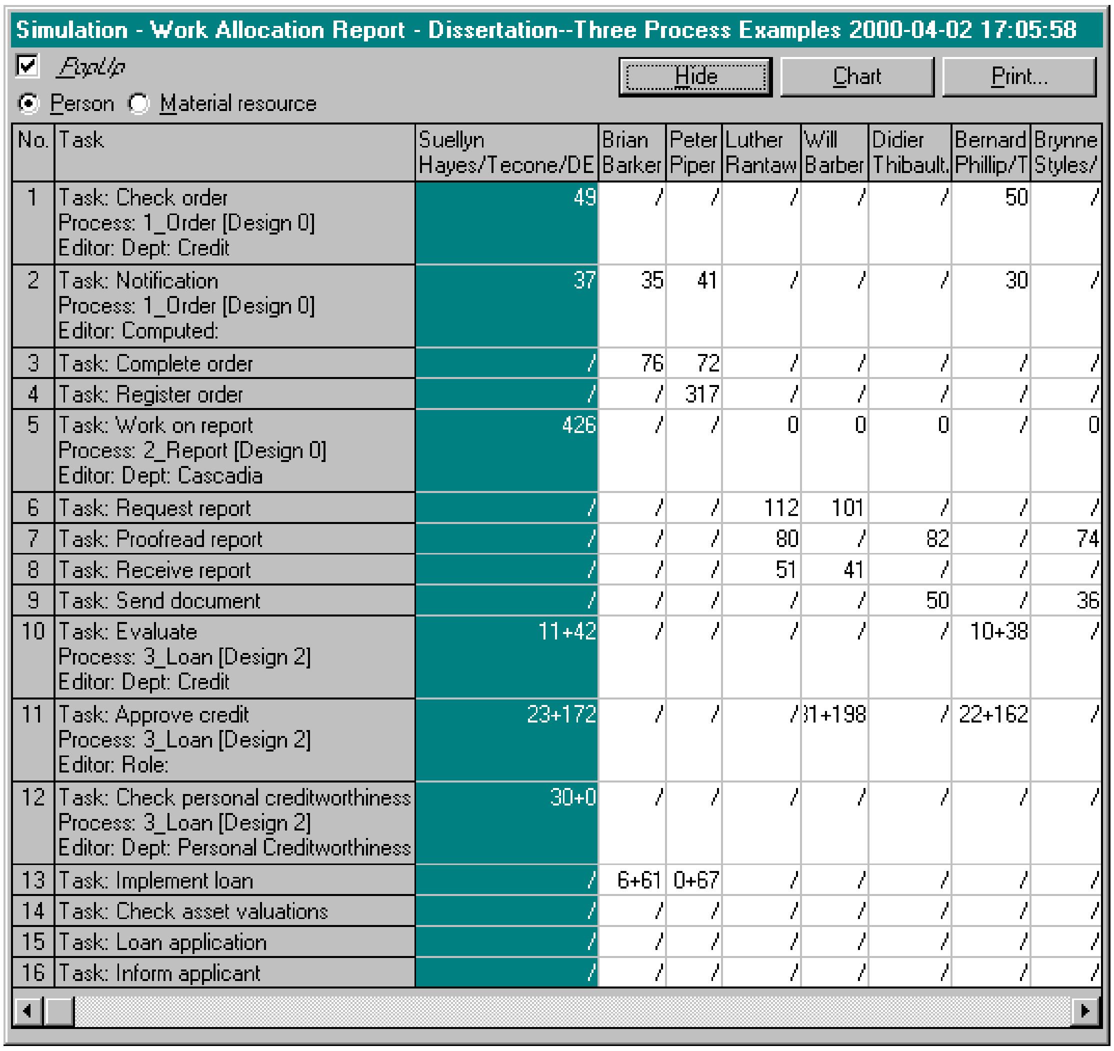
Figure 13-17. Resource Report—Chart

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 in Figure 13-17, we see that Suellyn Hayes might be one of the bottlenecks of the system. She has nearly 100% utilization rate, and a much longer waiting queue than any 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 process definitions could improve the system performance. For this purpose, the work allocation report can be used for further analysis.

13.3.5 Work Allocation Report

The work allocation report shown in Figure 13-18 presents among human or material resources the distribution of the simulated work items. Upon this


Simulation - Work Allocation Report - Dissertation--Three Process Examples 2000-04-02 17:05:58

☑ ExpUp

● Person ○ Material resource

No.	Task	Suellyn Hayes/Tecone/DE	Brian Barker	Peter Piper	Luther Rantaw	Will Barber	Didier Thibault,	Bernard Phillip/T	Brynne Styles/
1	Task: Check order Process: 1_Order [Design 0] Editor: Dept: Credit	49	/	/	/	/	/	50	/
2	Task: Notification Process: 1_Order [Design 0] Editor: Computed:	37	35	41	/	/	/	30	/
3	Task: Complete order	/	76	72	/	/	/	/	/
4	Task: Register order	/	/	317	/	/	/	/	/
5	Task: Work on report Process: 2_Report [Design 0] Editor: Dept: Cascadia	426	/	/	0	0	0	/	0
6	Task: Request report	/	/	/	112	101	/	/	/
7	Task: Proofread report	/	/	/	80	/	82	/	74
8	Task: Receive report	/	/	/	51	41	/	/	/
9	Task: Send document	/	/	/	/	/	50	/	36
10	Task: Evaluate Process: 3_Loan [Design 2] Editor: Dept: Credit	11+42	/	/	/	/	/	10+38	/
11	Task: Approve credit Process: 3_Loan [Design 2] Editor: Role:	23+172	/	/	/	31+198	/	22+162	/
12	Task: Check personal creditworthiness Process: 3_Loan [Design 2] Editor: Dept: Personal Creditworthiness	30+0	/	/	/	/	/	/	/
13	Task: Implement loan	/	6+61	0+67	/	/	/	/	/
14	Task: Check asset valuations	/	/	/	/	/	/	/	/
15	Task: Loan application	/	/	/	/	/	/	/	/
16	Task: Inform applicant	/	/	/	/	/	/	/	/

Figure 13-18. Work Allocation Report

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 (tasks). The data in the report can be displayed in chart form oriented to an activity (see Figure 13-20) or oriented to a human/material resource (see Figure 13-19) for ease of comparison and analysis.

It can be seen in Figure 13-19, Suellyn Hayes has participated in the work associated with all the three process definitions for executing activities of "Check order", "Notification", "Work on report", "Evaluate", "Approve credit" and "Check personal creditworthiness". Obviously, Suellyn Hayes has done more work for activity "Work on report" than any other activities. To prevent her from executing the activity can reduce her work.

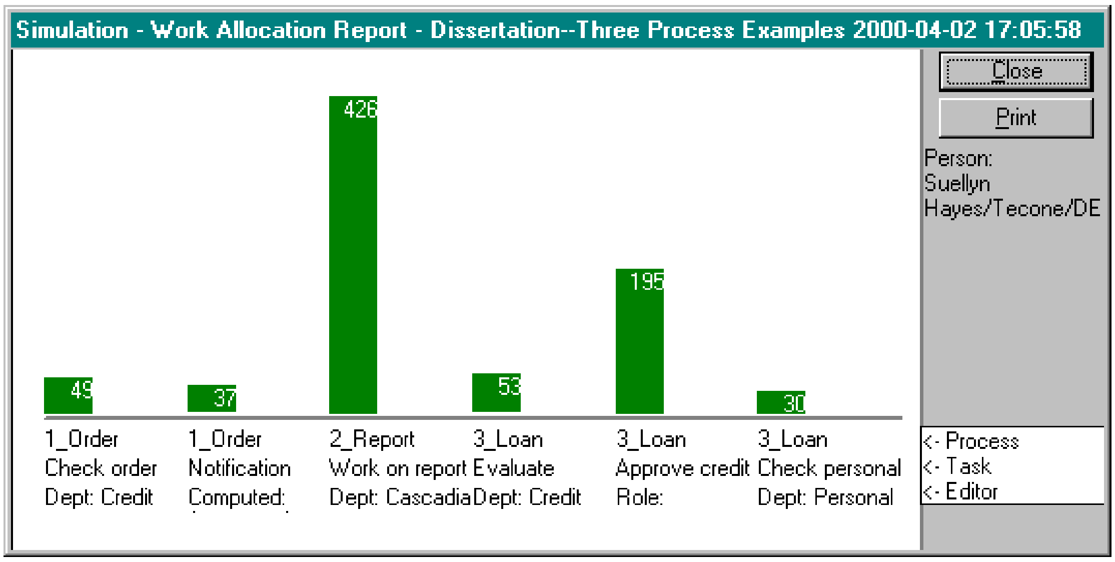


Figure 13-19. Work Allocation Report—Resource-Oriented

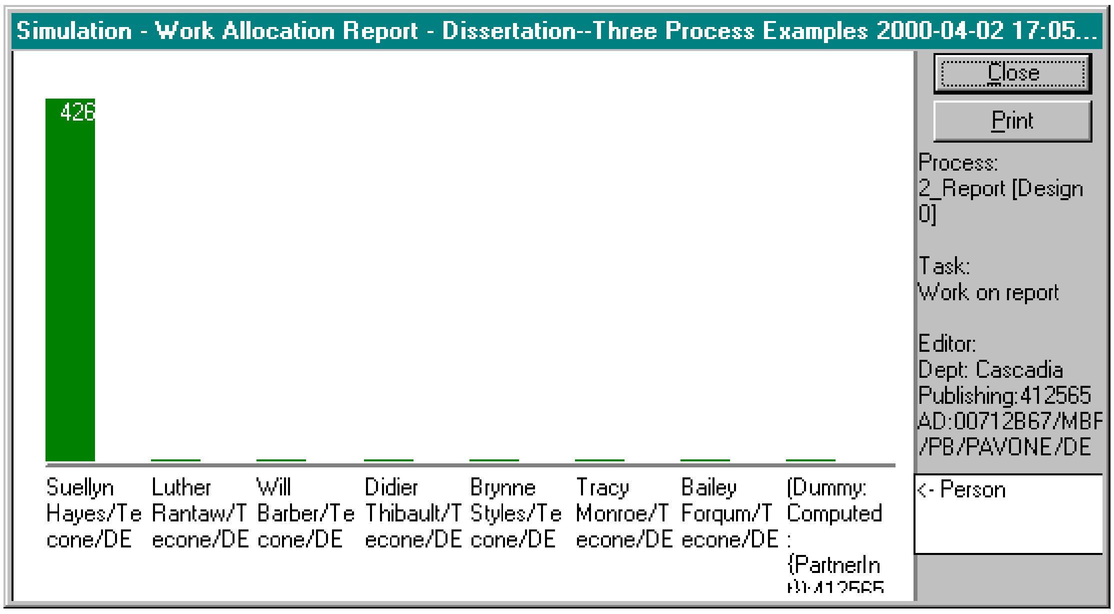


Figure 13-20. Work Allocation Report—Activity-Oriented

Figure 13-20 presents graphically how activity "Work on report" has been executed by all team members of department "Cascadia"—Suellyn Hayes has executed all 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. Must he complete the activity? Has he a 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 many activities within the analyzed process definitions. The system performance might be improved through

- assigning some activities to other editors,
- separating an activity with long executing time into two or more small activities and assigning them to different editors, or
- increasing members of the team editor in the organization and making most of the members have chance to execute the work associated with different activities—this has no effect, if all members of the team must complete the activities.

13.4 Events in the Espresso Simulation Model

The next-event approach is implemented in PM as the model algorithm for simulating process instances of multiple process definitions in different Espresso 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.

For each kind of events, a procedure is required to process the event: change state as well as statistic variables, and schedule new events and/or invoke conditional events.

13.4.1 Scheduled Events

According to the first occurrence time of the scheduled events kept in the eventlist, the simulator advances the clock, and the simulation run can then proceed with the event occurring at that time (see Section 8.2.2.2). 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 certain number of process creation events are scheduled for each start activity;

  2) when an arrival event happens at a start activity (i.e. 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 <u>Routing Work to Activity</u> and the work can be directly routed to a successor activity.

  The schedule of arrival events at a start activity (see Assumption 11-1 in Section 11.1.2) 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 (see algorithm for <u>Adding Work to Queue</u> in Section 12.5.1).
- *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, or
  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 stop a simulation run and generate the simulation reports. It is scheduled when a specified end activity is completed during animation. It is forced to occur,

  1) when the clock exceeds the given simulation period and not all running process instances should be terminated,
  2) when the eventlist is empty as the simulator advances the clock (at this time, the given simulation beginning/ending conditions have been met and all running process instances have been terminated), 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. Therefore, they do not need to be scheduled.

A work at an activity can be routed to a successor activity (i.e. 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 can be met when a participant depart event or a material release event happens.

A workflow participant can prepare to execute an activity (i.e. a *participant occupation event* occurs), if he is idle (or a dummy participant, or a Notes agent), and there is a work in his waiting queue with that 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 execute requires a delay and the delay time has not elapsed, the activity cannot 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 (i.e. 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 situation is released during detecting deadlocks (i.e. 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 (i.e. the algorithm for Routing Work to Activity is called); or
- when a parallel work is terminated at an end activity of the process definition—this could make another work no more parallel.

A material resource begins to be used (i.e. 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.

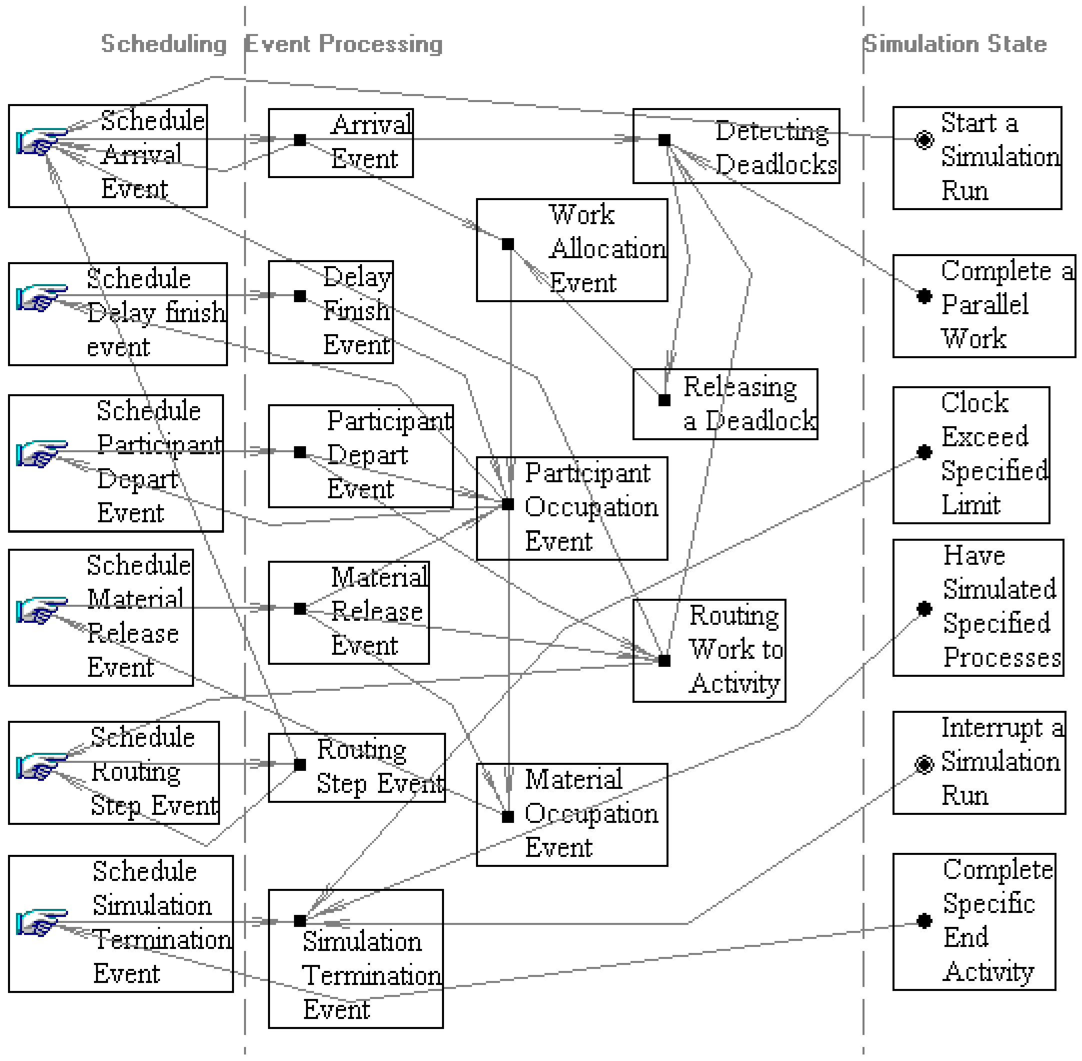

Scheduling
Event Processing
Simulation State
Schedule Arrival Event
Schedule Delay finish event
Schedule Participant Depart Event
Schedule Material Release Event
Schedule Routing Step Event
Schedule Simulation Termination Event
Arrival Event
Delay Finish Event
Participant Depart Event
Material Release Event
Routing Step Event
Simulation Termination Event
Work Allocation Event
Participant Occupation Event
Material Occupation Event
Detecting Deadlocks
Releasing a Deadlock
Routing Work to Activity
Start a Simulation Run
Complete a Parallel Work
Clock Exceed Specified Limit
Have Simulated Specified Processes
Interrupt a Simulation Run
Complete Specific End Activity

Figure 13-21. Events and Relevant Algorithms and States

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 allow the system analyst to interactively simulate a process instance from a specified start activity up to an end activities or a specified activity of the process definition. Simulating the process instances of one or all opened process definitions simultaneously in different Espresso application databases let the Espresso WfMS be simulated as it is constructed in the real world.

Various simulation settings are required for a simulation run. Some are defined with the tasks, links and the organization model; some are specified particularly for the simulation study to a process definition, and to a PAVONE Organization Database or a Notes Organization Directory. In addition, before a simulation run, the system analyst decides in the experiment settings the unit of the clock, period of a simulation run, protocol-generating intervals, and the Espresso application databases as well as configured PAVONE Organization Databases and/or Notes Organization Directories for each analyzed process definition. The experiment settings can be saved as a scenario for later simulation runs. The view settings for graphical display of the simulated system can be altered during the simulation run.

Dynamic process protocols can be generated during a simulation run and the statistical reports of the summary report, the activity report, the resource report and the work allocation report can be generated after the simulation run. The process protocol presents in detail an execution thread of a process instance from the start activity up to the specified (during simulation) or an end activity (during animation). 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 presents the distribution of work items associated with various activities, among simulated human or material resources. With the help of simulation reports, the system analyst can study the performance of an Espresso WfMS and detect bottlenecks in the system, so that he can 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 simulator advances the clock, or a simulation termination event occurs.

13.6 Further Work*

The following features 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 run 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 analyzed process definition.

- Multi-values of a variable within a distribution interval.
  In the current Espresso simulation model, the assignment of multi-values to a variable allows only one value getting from each subinterval of the empirical distribution function followed by the variable. 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 PAVONE Organization Databases and Notes Organization Directories) and personal holidays should be considered in the simulation model and could be treated in the same way. If a person begins executing an activity, the time taken for executing the activity will be prolonged if his personal holidays or regional public holidays occur before he completes the activity.

  If a resource is part-time engaged, the distribution of working time within a week, for example, could also be specified and simulated, not always uniformly distributed.

- Stand-in simulation.
  If personal holidays could be considered in the simulation model, stand-in might also be accounted for in the simulation study. Then the rules for a stand-in to take over a work should be given.

- 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.

- Synchronous execution of a team work.
  In the real world WfMS, it can happen that an activity (e.g. a meeting) must be executed by all the workflow participants of a team at the same time. If this could be simulated in the future, whether the material resources should be used for each individual workflow participant or for the whole team should also be specified and could be simulated.

- Empirical distribution for input of the simulation model.
  First the distribution function should be possible to be specified to a single time variable (e.g. the execution time of a certain activity). The time variable could then be specified to follow an empirical distribution.

- More queuing rules.
  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 a due date and a priority can be specified to a process instance in the Espresso WfMS.

  Further, if there are multiple process definitions implemented in a WfMS, the business processes associated with some process definitions are more important than those of other process definitions. Therefore, the simulation model should allow specifying priority to a process definition, and could simulate it.

  Although the worklist in the Espresso WfMS is sorted following 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 should 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 the order in which the queues are selected, the number of work items served at each polling session, and the time in transferring service among the queues (see [Mittra, 1986, p. 187]).

- Change of input during a simulation run.
  In the current simulation model, the input variables 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 points in time. So the input variable should be able to change .

- Input in the scenario.
  In addition to the experimental settings that can be saved as a scenario, other input variables of a simulation run 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 could be graphically presented, such as the plot of a queue length shown in Figure 8-4. This request might be specified with the experimental settings.

- Reproducibility.
  For comparison of alternative operating policies, the simulator should allow the system analyst to specify a seed before a simulation run, in order to generate a certain sequence of random numbers.

- Events for a non-stop activity.
  If a work is not allowed to be stopped at an activity, the simulator will try repeatedly to (let) choose the “Multiple Choice” links or to determine the values of variables in the “Condition” links. This does not actually coincide the real world WfMS. An event might be added in the simulation model for checking as scheduled the conditions, and another for choosing again the links just like the workflow engine does.

- Confidence interval of the simulation result.
  Some statistical system performance criteria, such as the average process duration, might be offered with a confidence interval (see Example 14-2 and Example 14-5 in Section 14.4).

- Design, simulation and comparison of alternative operating policies.
  This feature could ease making decision for improving a WfMS.

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 the 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 for estimation of the input variables to the simulation model as well.

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 simulated can be decided via analyzing the statistically significant difference between the means of the performance index of the system for any two alternatives.

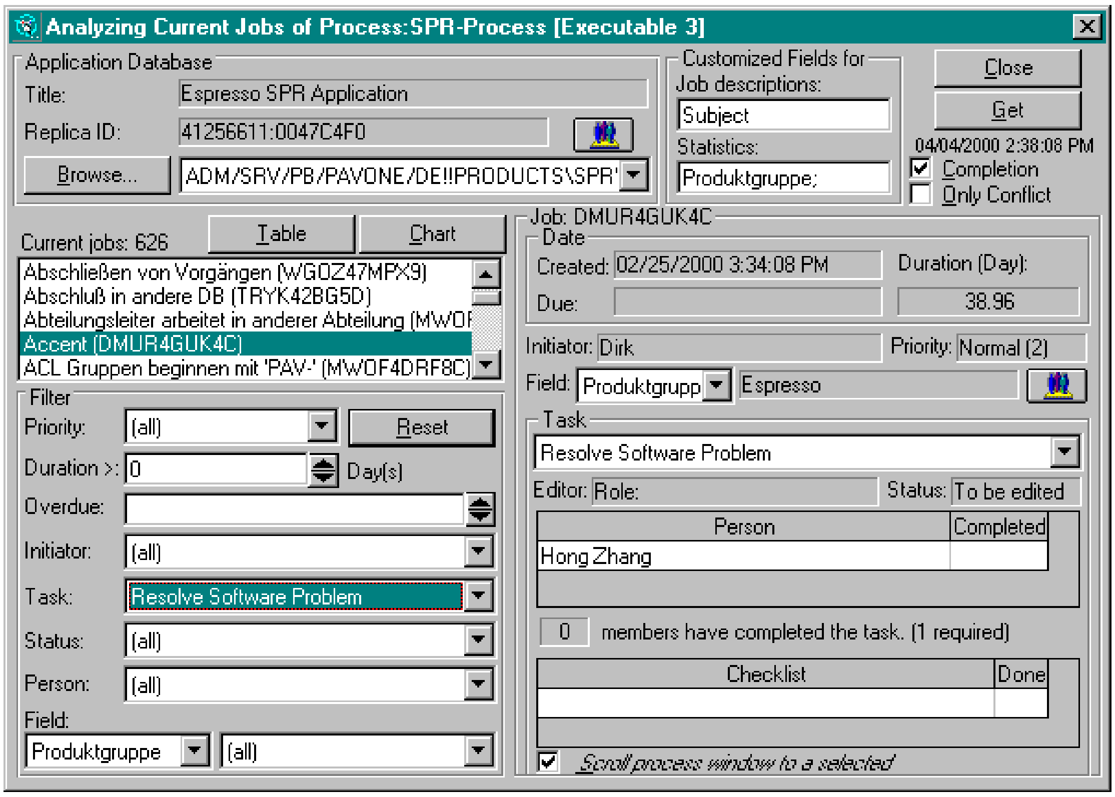
14.1 History Data Collection

“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 the Espresso WfMS, numerous 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, hours, or days) appropriate to the system has been previously established. 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 system 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 that 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 (jobs)



Analyzing Current Jobs of Process:SPR-Process [Executable 3]

Application Database
Title: Espresso SPR Application
Replica ID: 41256611:0047C4F0
ADM/SRV/PB/PAVONE/DE!!PRODUCTS\SPR'

Customized Fields for Job descriptions: Subject
Statistics: Produktgruppe;

04/04/2000 2:38:08 PM
☑ Completion
☐ Only Conflict

Current jobs: 626
Abschließen von Vorgängen (WG0Z47MPX9)
Abschluß in andere DB (TRYK42BG5D)
Abteilungsleiter arbeitet in anderer Abteilung (MWOF
Accent (DMUR4GUK4C)
ACL Gruppen beginnen mit 'PAV-' (MWOF4DRF8C)

Filter
Priority: (all)
Duration >: 0 Day(s)
Overdue:
Initiator: (all)
Task: Resolve Software Problem
Status: (all)
Person: (all)
Field: Produktgruppe (all)

Job: DMUR4GUK4C
Date
Created: 02/25/2000 3:34:08 PM Duration (Day):
Due: 38.96
Initiator: Dirk Priority: Normal (2)
Field: Produktgrupp Espresso
Task
Resolve Software Problem
Editor: Role: Status: To be edited

Person	Completed
Hong Zhang	

0 members have completed the task. (1 required)

Checklist	Done

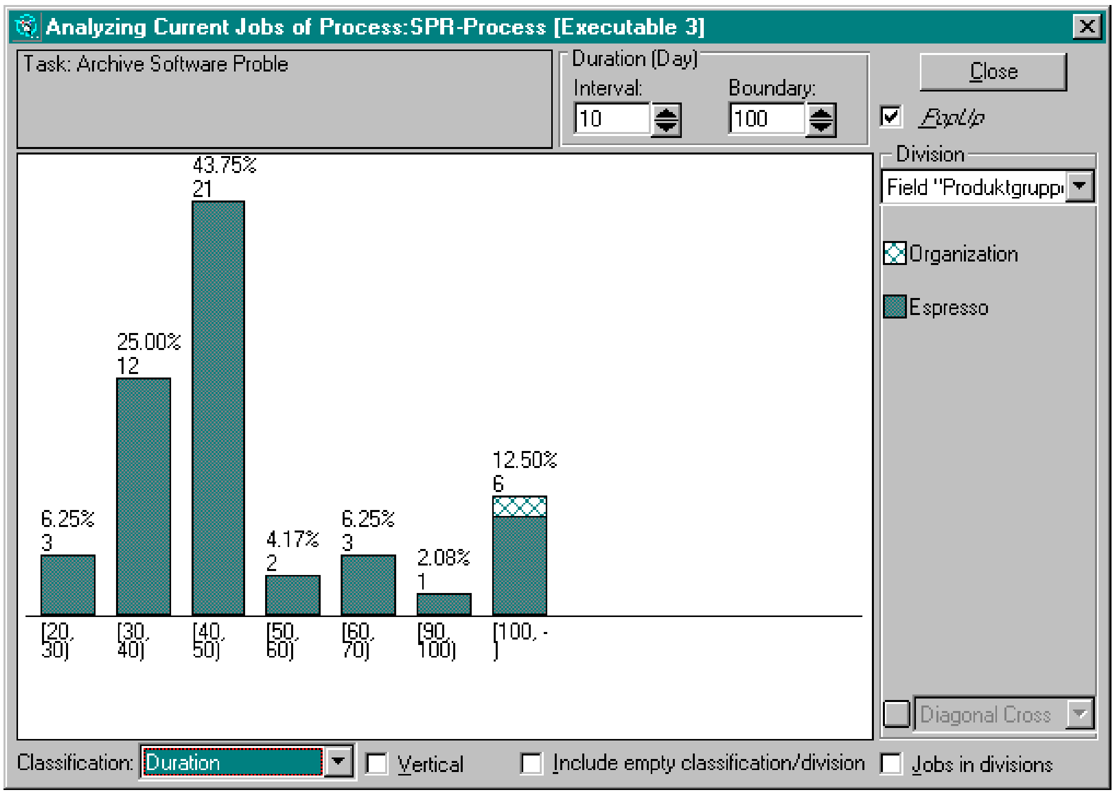
☑ Scroll process window to a selected

Figure 14-1. Process Instances Associated with a Process Definition

in accordance with a process definition. The most useful data can be gathered by the analyzer are:

- the duration of a completed process instance;
- the creation time of a process instance;
- the number of running works associated with the activities (tasks) within the process definition;
- the length of the worklist of a 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


Analyzing Current Jobs of Process:SPR-Process [Executable 3]

Task: Archive Software Proble

Duration (Day) Interval: 10 Boundary: 100

☑ FlipUp

Division: Field "Produktgrupp

▨ Organization

🟩 Espresso

Classification: Duration

	[20, 30)	[30, 40)	[40, 50)	[50, 60)	[60, 70)	[90, 100)	[100, -)
%	6.25%	25.00%	43.75%	4.17%	6.25%	2.08%	12.50%
Count	3	12	21	2	3	1	6

Figure 14-2. Collect Duration of Completed Process Instances

to determine some input variables of the WfMS that will be simulated in the future, or to validate the simulation model integrated in 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/Lieberman, 1974]

If the behavior of an element cannot 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/Lieberman, 1974, p. 625].

One question that may arise when choosing probability distributions for the simulation model is whether to use frequency distributions of historical data or to seek the theoretical probability distribution which best fits these data. The latter 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/Lieberman, 1974, p. 625].

The choice of a distribution function is sometimes troublesome to the beginning practitioner. Therefore, some guidelines should be helpful. Four considerations in the choice of a distribution function for a random variable should be indicated (see [Gottfried, 1984, p. 102]):

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, and
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, 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 χ^2 test gives 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.

"Some applications require certain specialized distribution functions that cannot easily be fitted to the available data. Computerized curve-fitting techniques are required in such situations." [Gottfried, 1984, p. 103] For some information on regression analysis, see [Graybeal/Pooch, 1980, pp. 56-59], [Naylor, 1969, pp. 123-131] and [Chorafas, 1965, pp. 162-165].

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 and cause-and-effect relationships (definitions and assumptions). 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. Thus, 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 could discriminate between the two sets of data. See [Hillier/Lieberman, 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 PM, the system analyst can design a corresponding process definition. To compare the simulation results of the process definition with the theoretical predictions can validate the Espresso simulation model integrated in PM.
- 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. Therefore, 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 Espresso simulation model integrated in PM by comparing with the simulation results.

The behavior of a WfMS resembles a queuing or waiting line problem. In a 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.

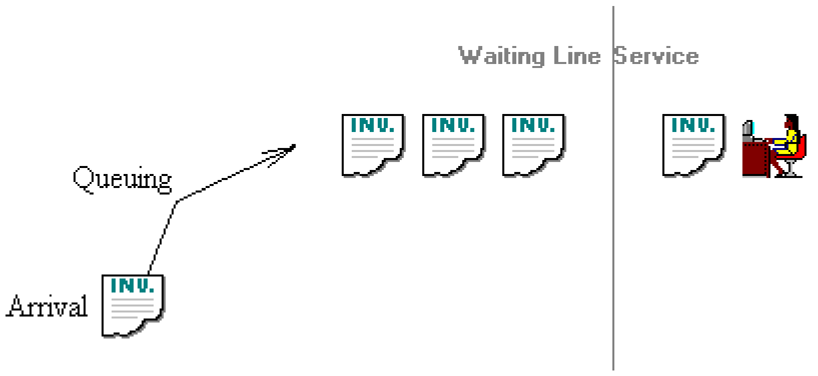


Figure 14-3. Single-channel Single-station Queue System

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 here. For more discussions about various queue models see [Tijms, 1986] and [Kleinrock/Gail, 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 λ and the service times are exponentially distributed with mean μ. 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 more 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 λ as before, but the service times are governed by a two-parameter distribution having mean μ and standard deviation σ, 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 14-1. Validate the Simulation Model
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,

$$\lambda = 30, \mu = 20, \text{ and } \sigma = 5$$

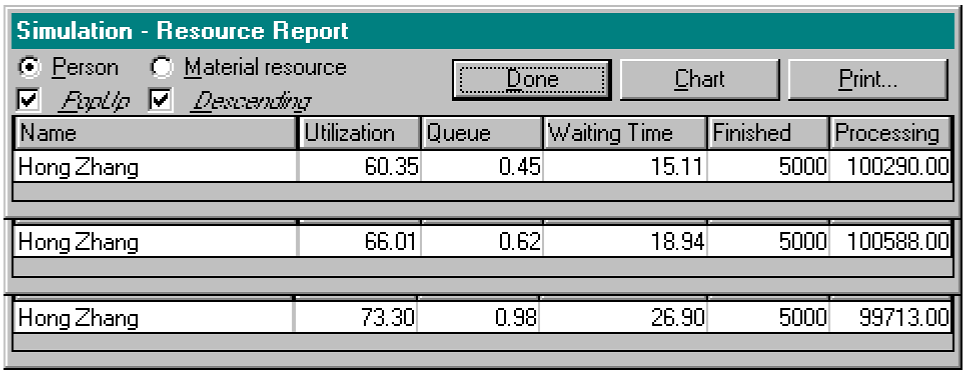
So

$$\beta = \mu/\lambda = 20/30 = 0.667$$

The theoretically expected values of the system can then be determined as

Expected waiting time = $(\sigma^2/\lambda + \beta^2\lambda)/(2(1-\beta))$
$= ((5)^2/(30) + (0.667)^2(30))/(2(1-0.667))$
$= 21.3$ (minutes)
Expected queue length = $(\sigma^2/\lambda^2 + \beta^2)/(2(1-\beta))$
$= ((5)^{2/}(30)^2 + (0.667)^2)/(2(1-0.667))$
$= 0.71$ (arrivals)
Expected fraction of time the server is idle = $(1-\beta)$
$= (1-0.667)$
$= 0.333$

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 minutes. 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 with the Espresso simulation model three times and get the results as shown in Figure 14-4.



[✓ Person] Material resource
[✓] PopUp [✓] Descending

Name	Utilization	Queue	Waiting Time	Finished	Processing
Hong Zhang	60.35	0.45	15.11	5000	100290.00
Hong Zhang	66.01	0.62	18.94	5000	100588.00
Hong Zhang	73.30	0.98	26.90	5000	99713.00

Figure 14-4. Simulation Results of a M/G/1 Queue

The simulation results are summarized in the following table for comparing with the theoretical predictions:

	Theoretical Prediction	Run 1 Results	Run 2 Results	Run 3 Results
Number of simulated process instances	(∞)	5000	5000	5000
Mean waiting time (minutes)	21.3	15.11	18.94	26.90
Mean queue length	0.71	0.45	0.62	0.98
Fraction of time the server is idle	0.333	0.3965	0.3399	0.2670

The agreement appears reasonable, so it enhances our confidence in the simulated results and the simulation model. To accept the simulation model at a confidence level, we might use *N*(0, 1) test discussed in Section 9.4.3. For this purpose, other 27 runs of simulating 5000 process instances are needed, so that we have at least 30 data to use this test to assess the validity of the mean waiting time and others generated by 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 an approximations subject to statistical error." [Mittra, 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 cannot 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:

$\overline{Y}$ = the calculated mean value (i.e. the sample mean) of the system performance criterion;

s = the calculated standard deviation of the system performance criterion;

n = the number of simulated values of the performance criterion used to calculated $\overline{Y}$ and s;

μ = the true mean value of the system performance criterion, which is unknown.

Here

$$\overline{Y} = \sum_{i=1}^{n} Y_i$$

and

$$s^2 = (\frac{1}{n}\sum_{i=1}^{n} Y_i^2) - (\overline{Y})^2$$

If the calculated Y_i, $i = 1, 2, ..., n$, is normally distributed, then the statistic

$$(\frac{\overline{Y}-\mu}{s})\sqrt{n-1}$$

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. So (See [Fishman, 1973, pp. 263-268])

$$\overline{Y}-t_{n-1,1-\alpha/2}s/\sqrt{n-1}<\mu<\overline{Y}+t_{n-1,1-\alpha/2}s/\sqrt{n-1}$$

Here $(1-\alpha)$ is the corresponding confidence level.

Example 14-2. Evaluate Simulation Results

100 process instances in accordance with the process definition in the Figure 1-3 in Section 1.3 was simulated with the specification that link "Order Accepted" has 100% routing probability (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 of the first activity to the end of the last activity will be evaluated. After the simulation run, the average duration $\overline{Y}$ was 1168.51 minutes with s = 60.38.

Determine the range of process duration μ 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 *t*-statistic from Table 1 in appendices, using linear interpolation between the tabulated values for

$$t_{60,\,0.975}=2.00$$

and

$$t_{120,\,0.975}=1.98$$

Thus,

$$t_{99,0.975}=2.00+(\frac{99-60}{120-60})(1.98-2.00)=1.987$$

The range of μ can now be determined as follows:

$$\overline{Y}\pm t_{99,\;0.975}\;s\;/\;\sqrt{n-1}\;=1168.51\pm(1.987)(60.38)\;/\;\sqrt{99}$$

$$=1168.51\pm 12.06$$

Therefore,

$$1156.45 < \mu < 1180.57$$

We conclude that the true mean value for the duration of a process instance Y falls between 1156.45 minutes and 1180.57 minutes at a 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 μ 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

$$\overline{Y} \pm Z_{0.5-\alpha/2}\ s\ /\ \sqrt{n-1}$$

when $n \geq 30$. Therefore, if the desired confidence interval is expressed as $\overline{Y} \pm \theta$, then

$$Z_{0.5-\alpha/2}\ s\ /\ \sqrt{n-1} = \theta$$

Solving for n, we obtain

$$n = (Z_{0.5-\alpha/2}\, s/\theta)^2 + 1$$

Thus, given the interval bound θ the least sample size n (>30) can be determined, so that the mean μ falls within the interval $\overline{Y} \pm \theta$ at a specified confidence level $100(1 - \alpha)\%$.

Example 14-3. Sample Sizes

We simulate the process instances of the process definition described in Example 14-2 with different values of sample size n (=10, 100, 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:

	$n = 10$		$n = 100$		$n = 1000$	
	$\overline{Y}$	s	$\overline{Y}$	s	$\overline{Y}$	s
Run 1	1147.90	58.70	1161.59	58.95	1162.15	64.55
Run 2	1145.10	58.19	1170.17	66.98	1156.94	57.10

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 14-4. Determine Sample Size

Again consider the duration problem described in Example 14-2. We wish to simulate a large enough number n of process instances so that the true mean value μ of duration Y falls within ±0.5% of the calculated mean $\overline{Y}$, at a 95% confidence level.

From Example 14-2, we have already seen that n must exceed 100, because we obtained a confidence interval 1168.51±12.06 when $n = 100$, and interval bound ±12.06 corresponds to ±1.03% of the calculated mean $\overline{Y}$ (= 1168.51). (±1.03% is larger than the desired ±0.5%.)

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

$$\theta = 0.005\overline{Y} = (0.005)(1160) = 5.8$$

Hence

$$n = (Z_{0.5-\alpha/2}\, s/\theta)^2 + 1 = (1.96 \times 60/5.8)^2 + 1 = 412$$

So when more than 412 process instances are simulated, we can say at a 95% confidence level that the duration falls within ±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 $\overline{Y}$ 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 (i.e. *m* blocks), where each block contains *n* simulated values of the performance criterion. Let

$\overline{Y}_i$ = the calculated mean value of the system performance criterion

for block *i*, *i* = 1, 2, ..., *m*. (Hence, each $\overline{Y}_i$ will be averaged over n_i independently generated values of the performance criterion.)

$\overline{Y}$ = the average of the calculated block means. Thus,

$$\overline{Y} = \frac{1}{m}\sum_{i=1}^{m}\overline{Y}_i$$

d = the standard deviation of the calculated block means about the average. That is,

$$d^2 = \left(\frac{1}{m}\sum_{i=1}^{m}\overline{Y}_i^2\right) - (\overline{Y})^2$$

μ = the true mean value (unknown) of the system performance criterion.

We can again establish a confidence interval for μ using the *t*-distribution. Now, however, the appropriate statistic for a given confidence lever (1 – α) is

$$-t_{m-1,1-\alpha/2} < \left(\frac{\overline{Y}-\mu}{d}\right) < t_{m-1,1-\alpha/2}$$

or

$$(\overline{Y} - t_{m-1,1-\alpha/2}d) < \mu < (\overline{Y} + t_{m-1,1-\alpha/2}d)$$

This equation states that the true mean μ falls within the interval

$$\overline{Y} \pm t_{m-1,1-\alpha/2}d$$

at a 100(1 – α)% 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

$$\overline{Y} \pm t_{n-1,1-\alpha/2} s / \sqrt{n-1}$$

This is the reason for favoring a blocking procedure. It should be understood, however, that the number of blocks *m* would 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 14-5. Blocking

The process definition in Example 14-3 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:

Block Number	Average Duration (minutes)
1	1161,59
2	1170,17
3	1141.61
4	1162.70
5	1159.21
6	1153.67
7	1163.05
8	1154.41
9	1158.35
10	1168.35

Determine the limits of μ that correspond to a 95% confidence level (where μ 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

$$\overline{Y} = \frac{1}{m}\sum_{i=1}^{m}\overline{Y}_i$$
$$= (1161.59 + 1170.17 + 1141.61 + 1162.70 + 1159.21 + 1153.67 + 1163.05 + 1154.41 + 1158.35 + 1168.35)/10$$
$$= 1159.311$$

So

$$d^2 = (\frac{1}{m}\sum_{i=1}^{m}\overline{Y}_i^2) - (\overline{Y})^2$$
$$= (1161.59^2 + 1170.17^2 + 1141.61^2 + 1162.70^2 + 1159.21^2 + 1153.67^2 + 1163.05^2 + 1154.41^2 + 1158.35^2 + 1168.35^2)/10 - 1159.311^2$$
$$= 60.038$$

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 1 in appendices as $t_{9,\,0.975} = 2.26$. We can now obtain the desired limits as

$$\bar{Y} \pm t_{m-1,1-\alpha/2}d$$
$$= 1159.311 \pm (2.26)\,(7.748) = 1159.311 \pm 17.510$$

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 $\bar{Y}_i$, $\bar{Y}$, 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 σ^2/n, where σ represents the true standard deviation of the random variates. As a rule, σ will be unknown. We can estimate σ, however, by utilizing the expression $d^2 = \sigma^2/n$. To do so, we write

$$\sigma^2 = n_1 d_1^2 = n_2 d_2^2$$

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 $\overline{Y} \pm \theta$ is the desired confidence interval. Now the above equation can be solved directly for d_2, and n_2 can then be obtained as

$$n_2 = n_1 d_1^2 / d_2^2$$

<u>Example 14-6. Determine Block Size</u>
Again consider the situation described in previous example. Determine how many process instances within each block should be simulated so that the true mean value falls within ±5% of the calculated mean of the process durations at a 95% confidence level.

If we base our calculations on the previously determined sample mean ($\overline{Y} = 1159.311$), then

$$\theta = 0.05\overline{Y} = 0.05\,(1159.311) = 57.96555$$

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,

$$n_2 = n_1 d_1^2 / d_2^2 = 100\,(7.748)^2/(25.648)^2 = 9.126$$

Thus, we conclude that the desired confidence interval can be obtained, if at least 10 (> 9.126) process instances are simulated for each of the ten blocks.

14.5 Conclusion

Collected data can be used to estimate the input variables of a simulation model and to validate the simulation model. Therefore, before using the simulation model, the system analyst should gather as much relevant data as possible. The analyzer integrated in PM can help in recording some data of an existing Espresso WfMS.

The input variables of process intercreation time, activity execution/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 chose 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 should be considered. If a set of empirical data of a random variable exists, it can be fitted via the χ^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 can be used.

Because a WfMS is a stochastic system, the results of a simulation run are also stochastic and cannot 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. Given the confidence interval, 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.

SUMMARY

The ProcessModeler (PM) is a graphical modeling tool for simple and easy design of flexible process definitions for the Espresso WfMS. It can verify the structure of a process definition as well as identify integrity errors. The join activities for merging parallel routing threads of a process instance are determined automatically by PM according to the activity network of the process definition. Work items waiting at the join activities within a process instance may yield a deadlock situation. Therefore, PM detects potential deadlock situations and assigns a priority to each join activity where such a situation could arise. The join priorities are used by the workflow engine to release deadlocks as they arise at run-time.

The simulator integrated in PM allows the user to visually simulate process definitions directly after modeling. The user can animate a process instance to see in detail how it can be created and routed in accordance the process definition, from activity to activity and from person to person. Simulating the Espresso WfMS, the user can estimate the system performance (duration, costs, bottlenecks, etc.) from a variety of generated reports.

Some of the statistical theory and methods introduced in this work can help the system analyst to choose the appropriate input for a simulation run and to evaluate the output. Others parts of it were used in developing the simulator.

The algorithms and assumptions described in this work are the basis of PM's capabilities for process design, verification and simulation.

REFERENCES

[Cacutalua, 1994]
Ndombe Cacutalua: “On Deadlocks in Concurrent Systems: A Petri Net based Approach for Deadlock Prediction and Avoidance”, 1994, R. Oldenbourg Verlag, München/Wien

[Chorafas, 1965]
Dimitris N. Chorafas: “Systems and Simulation”, 1965, Academic Press Inc., New York, NY, the United States of America

[Currid, 1994]
Cheryl Currid: “Reengineering ToolKit: 15 Tools and Technologies for Reengineering Your Organization”, 1994, Prima Publishing

[Dekker, 1977]
L. Dekker: “Simulation of Systems”, 1977, North-Holland Publishing Company, Amsterdam

[Eiselt, 1977]
Horst A. Eiselt (Helmut von Frajer): “Operations Research Handbook: Standard Algorithms and Methods with Examples”, 1977, Walter de Gruyter & Co., Berlin, Germany

[Fishman, 1973]
George S. Fishman: “Concepts and Methods in Discrete Event Digital Simulation”, 1973, John Wiley & Sons, Inc., New York, the United States of America

[Gerhards, 1991]
Gerhard Gerhards: “Seminar-, Diplom- und Doktorarbeit: Muster und Empfehlungen zur Gestaltung von Rechts- und wirtschaftswissenschaftlichen Prüfungsarbeiten”, 1991, Verlag Paul Haupt Bern und Stuttgart, Germany

[Gordon, 1978]
Geoffrey Gordon: “System Simulation”, 1978, 1969, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, the United States of America

[Gottfried, 1984]
Byron S. Gottfried: “Elements of Stochastic Process Simulation”, 1984, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, the United States of America

[Graybeal/Pooch, 1980]
Wayne J. Graybeal, Udo W. Pooch: “Simulation: Principles and Methods”, 1980, Winthrop Publishers, Inc., Cambridge, Massachusetts, the United States of America

[Gribik/Kortanek, 1985]
Raul R. Gribik, Kenneth O. Kortanek: “Extremal Methods of Operations Research,” 1985, Marcel Dekker, Inc., New York, the United States of America

[Harris, 1978]
Robert B. Harris: “Precedence and Arrow Networking Techniques for Construction”, 1978, John Wiley & Sons Inc., the United States of America

[Green/Hartley/Maritsas/Powner/Rumsey/Walker, 1975]
D. H. Green, M. G. Hartley, D. G. Maritsas, E. T. Powner, A. F. Rumsey, M. J. Walker: “Digital Simulation Methods”, 1975, Peter Peregrinus Ltd., Huddersfield, England

[Hillier/Lieberman, 1974]
Frederick S. Hillier, Gerald J. Lieberman: “Operations Research”, 1974, Holden-Day, Inc., San Francisco, California, the United State of America

[Hilpert/Riempp/Nastansky, 1994]
Wolfgang Hilpert, Gerold Riempp, Ludwig Nastansky: “The GroupFlow System: Workflow Management in Distributed Organizations”, 1994, URL: http://pbfb5www.uni-paderborn.de/www/WI/WI2/wi2_lit.nsf, University of Paderborn, Germany

[Hollingsworth, 1995]
David Hollingsworth: “Workflow Management Coalition: The Workflow Reference Model”, 1995, URL: http://www.wfmc.org, Winchester, United Kingdom

[Hu, 1982]
T. C. Hu: “Combinatorial Algorithms”, 1982, Addison-Wesley Publishing Company, Inc., the United States of America

[Huth/Erdmann/Nastansky, 2001]
Carsten Huth, Ingo Erdmann, Ludwig Nastansky: “GroupProcess: Using Process Knowledge from the Participative Design and Practical Operation of Ad Hoc Processes for the Design of Structured Workflows”, 2001, URL: http://pbfb5www.uni-paderborn.de/www/WI/WI2/wi2_lit.nsf, University of Paderborn, Germany

[Kleinrock/Gail, 1996]
Leonard Kleinrock, Richard Gail: “Queueing Systems: Problems and Solutions”, 1996, John Wiley & Sons, Inc., New York, NY

[Kremer, 1999]
Rolf Kremer: “Replikatives Informationsmanagement in verteilten Groupware-Umgebungen: Entwicklung, Architekturdarstellung und ProotypöDesign des Replikationsmodells ‘GroupReplic’”, 1999, Shaker Verlag, Aachen, Germany

[Koulopoulos, 1995]
Thomas M. Koulopoulos: “The Workflow Imperative”, 1995, Van Nostrand Reinhold, New York

[Lawrence, 1997]
Peter Lawrence: “Workflow Handbook”, 1997, John Wiley & Sons Ltd, Chichester, Great Britain

[Lehman, 1977]
Richard S. Lehman: “Computer Simulation And Modeling: An Introduction”, 1977, Lawrence Erlbaum Associates, Inc., New Jersey, the United States of America

[Lewis/Smith, 1979]
T. G. Lewis, B. J. Smith: “Computer Principles of Modeling and Simulation”, 1979, Houghton Mifflin Company, Boston, the United States of America

[Lorange, 1993]
Peter Lorange: “Strategic Planning and Control: Issues n the Strategy Process”, 1993, Blackwell Publishers, Massachusetts, the United States of America

[Maisel/Gnugnoli, 1972]
Herbert Maisel, Giuliano Gnugnoli: “Simulation of Discrete Stochastic Systems”, 1972, Science Research Associates, Inc., Chicago, the United States of America

[Mittra, 1986]
Sitansu S. Mittra: “Decision Support Systems: Tools and Techniques”, 1986, John Wiley & Sons, Inc., Toronto, Canada

[Nastansky/Hilpert, 1994]
Ludwig Nastansky, Wolfgang Hilpert: “The GroupFlow System: A Scalable Approach to Workflow Management between Cooperation and Automation”, 1994, URL: http://pbfb5www.uni-paderborn.de/www/WI/WI2/wi2_lit.nsf, University of Paderborn, Germany

[Nastansky/Hilpert, 1996]
Ludwig Nastansky, Wolfgang Hilpert: “The GroupFlow Framework: Enterprise Model and Architecture of the Workflow System”, 1996, URL: http://pbfb5www.uni-paderborn.de/www/WI/WI2/wi2_lit.nsf, University of Paderborn, Germany

[Naylor, 1969]
Thomas H. Naylor: “The Design of Computer Simulation Experiments”, 1969, Duke University Press, Durham, N. C., the United States of America

[Naylor/Balintfy/Burdick/Chu, 1966]
Thomas H. Naylor, Joseph L. Balintfy, Donald S. Burdick, Kong Chu: “Computer Simulation Techniques”, 1966, John Wiley & Sons, Inc., New York, the United States of America

[Ott, 1994]
Marcus Ott: “Conceptual Design and Implementation of a Graphical Workflow-Modelling Editor in the Context of Distributed Groupware-Databases”, Master Thesis, 1994, URL: http://pbfb5www.uni-paderborn.de/www/WI/WI2/wi2_lit.nsf, University of Paderborn, Germany

[Ott, 1999]
Marcus Ott: “Organization Design as a Groupware-supported Team Process (GroupOrga—Participative and Distributed Organization Design for Office Information and Workflow Management Systems”, Dissertation, 1999, URL: http://pbfb5www.uni-paderborn.de/www/WI/WI2/wi2_lit.nsf, University of Paderborn, Germany

[Ott/Nastansky/Brockmeyer, 1996]
Marcus Ott, Ludwig Nastansky, Frank Brockmeyer: “A Groupware-based architecture for secure interaction of intranet databases and the internet”, 1996, URL: http://pbfb5www.uni-paderborn.de/www/WI/WI2/wi2_lit.nsf, University of Paderborn, Germany

[Riempp, 1998]
Gerold Riempp: “Wide Area Workflow Management”, 1998, Springer, London

[Rosko, 1972]
Joseph S. Rosko: “Digital Simulation of Physical Systems”, 1972, Addison-Wesley Publishing Company, Inc., the United States of America

[Solomon, 1983]
Susan L. Solomon: “Simulation of Waiting-line Systems”, 1983, Prentice-hall, Inc., Englewood Cliffs, New Jersey, The United States of America

[Steward, 1981]
Donald V. Steward: “Systems Analysis and Management: Structure, Strategy and design”, 1981, Petrocelli Books, inc., the United States

[Tijms, 1986]
Henk C. Tijms: “Stochastic Modelling and Analysis: A Computational Approach”, 1986, John Wiley & Sons Ltd., Chichester, Great Britain

[Toulemonde/Gabathuler/Jansen/Rossini/Wylie/Schaper, 1998]
Christophe Toulemonde, Jakob Gabathuler, Brendan Jansen, Laura Rossini, John Wylie, Lancelot Schaper: “Lotus Solutions for the Enterprise, Volume 5: NotesPump, The Enterprise Data Mover”, 1998, International Business Machines Corporation 1998

[Van der Aalst/ter Hofstede, 1998]
W.M.P. van der Aalst, A.H.M. ter Hofstede: “Verification of Workflow Task Structure: A Petri-net-based approach”, 1998, Research report 380, AIFB, University of Karlsruhe, Germany

[Weinberg, 1980]
Victor Weinberg: “Structured Analysis”, 1980, YOURDON inc., New York, N.Y.

[WfMC, 1996]
Workflow Management Coalition: “Terminology & Glossary”, 1996, URL: http://www.wfmc.org, Brussels, Belgium

[WfMC, 1998]
Workflow Management Coalition: “Workflow and Internet: Catalysts for Radical Change”, 1998, URL: http://www.wfmc.org, Winchester, United Kingdom

APPENDICES

Symbols in Definitions and Algorithms

{}:	to enclose constant elements of a set or collection;
ϕ:	an empty set (a set without element);
$\forall$:	for all;
∂:	for any;
$\exists$:	there is, or exist;
$\in$:	is an element of;
$\notin$:	is not an element of;
$\subset$:	is a subset of;
$\subseteq$:	is a subset or the same set of;
$\equiv$:	is congruent with ;
$\|A\|$:	the number of elements in set A, if A represents a set; or the absolute value of A, if A is a variable;
$A \leftarrow B$:	let A have the same value as that of B;
$A \cap B$:	the intersection of set A and set B;
$A \cup B$:	the union of set A and set B;
$A - B$:	to remove all elements of set B from set A, if A and B represent sets; or to subtract B from A, if A and B are variables;
INT(α) or [α]:	the largest integer in α or the 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 <u>xxx</u>:	to run a procedure with the underlined name (here "xxx").

DTD of XML for PAVONE Process Definition

```
<!-- PAVONEProcessDefinition Version 5.00/1000-------------------------------------->
<!ELEMENT PAVONEProcessDefinition (PDGeneral, ActivityDefinitions)>
<!-- PDGeneral ------------------------------------------------------------------------------>
<!ELEMENT PDGeneral (PDBasic, PDGraph (, PDEscalation)? (,
PDSimulation)? (, PDAuxiliary)?)>
<!-- PDGeneral/PDBasic -------------------------------------------------------------------->
<!ELEMENT PDBasic ((OriginalProcessDBPath,)? (ProcessName,)?
(ProcessDefinitionID, )? PDVersionID (, PDVersionNo)? (, PDStatus)? (,
PDDescription)?, PDCategories, PDRole)>
<!ELEMENT OriginalProcessDBPath Path>
<!ELEMENT ProcessName Name>
<!ELEMENT (ProcessDefinitionID | PDDescription) #PCDATA>
<!ELEMENT PDVersionNo #PCDATA>
<!ELEMENT PDCategories (PDCategory)* >
<!ELEMENT PDCategory #PCDATA>
<!ELEMENT PDRole (PDCreator, PDDesigners, PDSupervisors, PDServers)>
<!ELEMENT PDDesigners (PDDesigner)+>
<!ELEMENT PDSupervisors (PDSupervisor)+>
<!ELEMENT PDServers (PDServer)+>
<!ELEMENT (PDSupervisor | PDServers) (NotesRole | Name)>
<!ELEMENT (PDDesigner | PDDesigner) Name>
<!ELEMENT NotesRole [Name]>
<!ELEMENT PDStatus ('Build-time' | 'Run-time')>
<!-- PDGeneral/PDGraph ------------------------------------------------------------------->
<!ELEMENT PDGraph (PDSizeUnit, Workspace, ZoomMainMap, Grid,
LinkGraph, PDMessage, TextBoxes, ActivityGroupMaps)>
<!ELEMENT Workspace (Width, Height, PicturePath)>
<!ELEMENT PicturePath ('(None)' | '(Default)' | Path)>
<!ELEMENT Grid (Size, IsShow)>
<!ELEMENT LinkGraph (LinkArrow, IsGraphicRoutngOptions)>
<!ELEMENT LinkArrow(Angle, Length, Offset)>
<!ELEMENT IsGraphicRoutngOptions ('true' | 'false')>
<!ELEMENT PDMessage (Label, PopUp)>
<!ELEMENT Label (Width, IsAbbreviate,LabelActivity, LabelActivityGroup,
LabelLink )>
<!ELEMENT PopUp (PopUpActivity, PopUpLink)>
<!ELEMENT IsAbbreviate ('true' | 'false')>
<!ELEMENT LabelActivity (Font, ColorForInvalidActivity,
ColorForActivityBorder, ViewActivity)>
<!ELEMENT LabelActivityGroup Font>
<!ELEMENT LabelLink (IsAcrossLine, Font, ColorForActiveLink, ViewLink)>
<!ELEMENT IsAcrossLine ('true' | 'false')>
<!ELEMENT PopUpActivity ViewActivity>
<!ELEMENT PopUpLink ViewLink>
```

```
<!ELEMENT ViewActivity (IsShowName, IsShowAlias, IsShowNo,
IsShowMapNo, IsShowEditorName, IsShowEditorTeam, IsShowStandinName,
IsShowStandinTeam, IsShowObject, IsShowNotesColumnIconFormula,
IsShowCheckList, IsShowCheckListValidationSwitch, IsShowInstructions,
IsShowValidationFormula, IsShowLotusScriptCode, IsShowResources,
IsShowFixedCosts, IsShowMaximumDuration,
IsShowMaximumProcessingTime, IsShowMaximumActivityInstances,
IsShowProcessingTime)>
<!ELEMENT ViewLink (IsShowName, IsShowRoutingOption,
IsShowLotusScriptCode, IsShowMergeOption, IsShowMergeFields,
IsShowMergeMainDocument, IsShowMergeDeleteDocument,
IsShowMailNotification, IsShowTransportTime)>
<!ELEMENT (IsShowName | IsShowAlias  | IsShowNo  |  IsShowMapNo |
IsShowEditorName | IsShowEditorTeam | IsShowStandinName | IsShowStandinTeam |
IsShowObject | IsShowNotesColumnIconFormula | IsShowCheckList |
IsShowCheckListValidationSwitch| IsShowInstructions | IsShowValidationFormula |
IsShowLotusScriptCode | IsShowResources | IsShowFixedCosts |
IsShowMaximumDuration | IsShowMaximumProcessingTime |
IsShowMaximumActivityInstances | IsShowProcessingTime | IsShowRoutingOption |
IsShowLotusScriptCode | IsShowMergeOption | IsShowMergeFields |
IsShowMergeMainDocument | IsShowMergeDeleteDocument |
IsShowMailNotification | IsShowTransportTime) ('true' | 'false')>
<!ELEMENT TextBoxes (DefaultFont, (TextBox)* )>
<!ELEMENT TextBox (Contents, MapNo, Position, Font, TextboxLine)>
<!ELEMENT Contents #PCDATA>
<!ELEMENT TextboxLine(LineOption, LineStyle)>
<!ELEMENT LineOption ('none' | 'vertical' | 'horizontal')>
<!ELEMENT LineStyle ('solid' | 'dash' | 'dot' | 'dash-dot' | 'dash-dot-dot')>
<!ELEMENT ActivityGroupMaps (IconFile, (ActivityGroupMap)* )>
<!ELEMENT ActivityGroupMap (Name, MapNo, ParentMapNo, Zoom, Position,
LabelOffset)>
<!-- PDGeneral/PDEscalation ------------------------------------------------------------->
<!ELEMENT PDEscalation (MaximumInstances, EscalationPI)>
<!ELEMENT EscalationPI (MaximumDuration, MaximumIterations)>
<!—PDGeneral/PDSimulation ------------------------------------------------------------->
<!ELEMENT PDSimulation (SettingPath)>
<!—PDGeneral/PDAuxiliary-------------------------------------------------------------->
<!ELEMENT PDAuxiliary (BuildProcessModeler (, PDApplicationDatabase)? (,
PDDominoDirectory)?, PDOrganizationDatabase)>
<!ELEMENT BuildProcessModeler #PCDATA>
<!ELEMENT PDApplicationDatabase (DBSpecification, FormFilters,
AgentFilters)>
<!ELEMENT PDDominoDirectory (DBSpecification, GroupFilters)>
<!ELEMENT PDOrganizationDatabase DBSpecification>
<!ELEMENT DBSpecification (Path (, ReplicaID)? (, Title)? )>
<!ELEMENT (FormFilters | AgentFilters | GroupFilters) (Filter)* >
<!ELEMENT Filter#PCDATA>
```

```
<!-- ActivityDefinitions----------------------------------------------------------------------->
<!ELEMENT ActivityDefinitions(ActivityDefinition)+>
<!ELEMENT ActivityDefinition (ADBasic, ADGraph (, ADEscalation) (,
ADEnterprise)? (, ADSimulation) )>
<!-- ActivityDefinitions/ADBasic ------------------------------------------------------------>
<!ELEMENT ADBasic (ActivityNo, Name, Alias, IsStartProcess, Editor, Stand-in,
Object, Instructions, CheckList, CheckListValidationSwitch, ValidationFormula
(,IsLogicValidationFormula)? , LotusScriptCode, NotesColumnIconFormula,
JoinPriority, IsLogOnComplete, ADLinks)>
<!ELEMENT Alias ('' | #PCDATA) >
<!ELEMENT IsStartProcess  ('true' | 'false')>
<!-- Editor , Stand-in ------------------------------------------------------------------------>
<!ELEMENT (Editor | Stand-in) (OrganizationEntity, OrganizationEntityType,
TeamParameter, TeamType, Completer)>
<!ELEMENT OrganizationEntityType ( 'Person' | 'Role'  | 'Department'  | 'Workgroup'
| 'NotesGroup'  | 'NotesAgent'  | 'DynamicReadFromField'  | 'Default' )>
<!ELEMENT OrganizationEntity   ( ( FullName (';' FullName)*)? | RoleName|
DepartmentName | WorkgroupName | GroupName | AgentName | FieldName ) >
<!ELEMENT ( RoleName | DepartmentName | WorkgroupName | GroupName |
AgentName ) #PCDATA >
<!ELEMENT TeamParameter (Parameter | '#' FieldName '#')>
<!ELEMENT Parameter ('' | #PCDATA)>
<!ELEMENT TeamType ('0' | '1' | '2')>
<!-- TeamType=> 0: individual / 1: team (given members) / 2: team (given number of
members) ------------------------------------------------------------------------------------>
<!ELEMENT Completer ( #PCDATA | (FullName (';' FullName)*) ) >
<!-- If TeamType is 2 (a team with given number of members), Completer => -1: only
manager / 0: all members / otherwise: the number of members. -------------------------->
<!ELEMENT FullName#PCDATA >
<!--------------------------------------------------------------------------------------------->
<!ELEMENT Object #PCDATA>
<!ELEMENT (Instructions | CheckList) ('' | #PCDATA)>
<!ELEMENT CheckListValidationSwitch ('off' | 'on')>
<!ELEMENT (ValidationFormula | NotesColumnIconFormula) ('' | #PCDATA)>
<!ELEMENT (IsLogicValidationFormula| IsLogOnComplete) ('true' | 'false')>
<!ELEMENT JoinPriority #PCDATA>
<!--  ADLinks ----------------------------------------------------------------------------------->
<!ELEMENT ADLinks (ADLink)* >
<!ELEMENT ADLink (Destination, Name, RoutingOption, RoutingCondition,
MailNotification, LDMerge, LotusScriptCode, LDTransportTime, LDGraph)>
<!ELEMENT Destination (ActivityNo (, Name) )>
<!ELEMENT LotusScriptCode  ('' | #PCDATA)>
<!ELEMENT RoutingOption ('Always' | 'MultipleChoice' | 'ExclusiveChoice' |
'Conditional' | 'Else' )>
<!ELEMENT MailNotification (MailNotificationSwitch, Condition)>
<!ELEMENT  MailNotificationSwitch ('on' | 'off' | 'conditional' )>
<!ELEMENT (RoutingCondition | Condition) ('' | #PCDATA)>
```

```
<!ELEMENT LDMerge (MergeOption, IsMainDocument,
IsDeleteIfNotMainDocument, MergeFields)>
<!ELEMENT MergeOption ('auto' | 'manual')>
<!ELEMENT (IsMainDocument | IsDeleteIfNotMainDocument) ('true' | 'false')>
<!ELEMENT MergeFields (FieldName)+ >
<!ELEMENT LDGraph (IsCommonColor, ColorGiven, LinkInMaps, LinkSplit)>
<!ELEMENT IsCommonColor ('true' | 'false')>
<!ELEMENT LinkInMaps (LinkInMap)+ >
<!ELEMENT LinkInMap (MapNo, Position)
<!ELEMENT LinkSplit (ConnectionNumber, ToConnectionPointOffset,
FromConnectionPointOffset)>
<!ELEMENT ConnectionNumber #PCDATA>
<!ELEMENT (ToConnectionPointOffset | FromConnectionPointOffset) (X, Y)>
<!ELEMENT Radius #PCDATA>
<!-- ActivityDefinitions/ADGraph------------------------------------------------------------>
<!ELEMENT ADGraph (MapNo, IconFile, Position, LabelOffset,
ActivityInOtherMaps)>
<!ELEMENT ActivityInOtherMaps (ActivityInOtherMap)* >
<!ELEMENT ActivityInOtherMap (MapNo, Position)
<!-- ActivityDefinitions/ADEscalation ------------------------------------------------------>
<!ELEMENT  ADEscalation (MaximumInstances, EscalationAI)>
<!ELEMENT EscalationAI (MaximumDuration, MaximumProcessingTime)>
<!-- ActivityDefinitions/ADEnterprise ------------------------------------------------------>
<!ELEMENT ADEnterprise (ADRealTimeNotesActivity, ADMQSeries)>
<!ELEMENT ADRealTimeNotesActivity (IsRTNAActivate, RTNALinkTemplate,
RTNAActiveServer, RTNAOptions, RTNADestination,
RTNADestDatabase_Def, RTNADestMetadata, RTNADestKeyList,
RTNADestFieldList, RTNASrcKeyList, RTNASrcFieldList, RTNAOpenEvent,
RTNACreateEvent, RTNAUpdateEvent, RTNADeleteEvent,
RTNAStorageOptions, RTNAStaticFields)>
<!ELEMENT IsRTNAActivate ('true' | 'false')>
<!ELEMENT  RTNALinkTemplate #PCDATA>
<!ELEMENT (RTNAActiveServer | RTNADestination | RTNADestDatabase_Def |
RTNADestMetadata | RTNADestKeyList | RTNADestFieldList | RTNASrcKeyList |
RTNASrcFieldList | RTNAStaticFields) ('' | #PCDATA)>
<!ELEMENT RTNAOptions ('OpenFailCreate')? >
<!ELEMENT (RTNAOpenEvent | RTNACreateEvent | RTNAUpdateEvent |
RTNADeleteEvent) ('true' | 'false')>
<!ELEMENT RTNAStorageOptions ('DelAll' | 'KeepAll' | 'Select')>
<!ELEMENT ADMQSeries (IsMQSActivate, MQSManager, MQSAction,
MQSPutQueue, MQSPutMsg, MQSGetQueue, MQSGetValue)>
<!ELEMENT IsMQSActivate ('true' | 'false')>
<!ELEMENT (MQSManager | MQSPutQueue | MQSPutMsg | MQSGetQueue |
MQSGetValue) ('' | #PCDATA)>
<!ELEMENT MQSAction ('Put' | 'Get' | 'Request')>
<!-- ActivityDefinitions/ADSimulation ------------------------------------------------------>
<!ELEMENT ADSimulation (ADFixedCosts, WorkDivisionProbability,
IsOnlyCompleterToProcess, ADProcessingTime, ADDelayTime, ADMaterials)>
```

```
<!ELEMENT ADFixedCosts (Value, Currency)>
<!ELEMENT WorkDivisionProbability #PCDATA>
<!ELEMENT IsOnlyCompleterToProcess ('true' | 'false')>
<!ELEMENT ADMaterials (ADMaterial)*>
<!ELEMENT ADMaterial (MaterialName, ADMaterialTime, IsSynchronousStart,
IsSynchronousForward)>
<!ELEMENT MaterialName  #PCDATA>
<!ELEMENT (IsSynchronousStart | IsSynchronousForward) ('true' | 'false')>
<!-- Overall---------------------------------------------------------------------------->
<!ELEMENT  PDVersionID  #PCDATA>
<!ELEMENT IconFile #PCDATA>
<!ELEMENT Font | DefaultFont (Name, Size, IsBold, IsItalic, IsStrikethru,
IsUnderline, Color)>
<!ELEMENT (Color| ColorForInvalidActivityr| ColorForActivityBorderr|
ColorForActiveLink | ColorGiven) (NamedColor | SystemColor | RGBColor)>
<!ELEMENT NamedColor ('Black' | 'Silver' | 'Gray' | 'White' | 'Maroon' | 'Red' | 'Purple'
| 'Fuchsia' | 'Green' | 'Lime' | 'Olive' | 'Yellow' | 'Navy' | 'Blue' | 'Teal' | 'Aqua')>
<!ELEMENT SystemColor ('ScrollBars' | 'Desktop' | 'ActiveTitleBar' |
'InactiveTitleBar' | 'MenuBar' | 'WindowBackground' | 'WindowFrame' | 'MenuText' |
'WindowText' | 'TitleBarText' | 'ActiveBorder' | 'InactiveBorder' |
'ApplicationWorkspace' | 'Highlight'  | 'HighlightText'  | 'ButtonFace'  | 'ButtonShadow'
| 'GrayText'  | 'ButtonText'  | 'InactiveCaptionText'  | '3DHighlight'  | '3DDKShadow' |
'3DLight'  | 'InfoText'  | 'InfoBackground' )>
<!ELEMENT RGBColor ('#' #PCDATA) >
<!ELEMENT PDSizeUnit ('twip' | 'cm' | 'inch')>
<!ELEMENT (MaximumDuration| MaximumProcessingTime) (Value, TimeUnit)>
<!ELEMENT (LDTransportTime | ADProcessingTime | ADDelayTime |
ADMaterialTime) (Value, TimeUnit, Deviation)>
<!ELEMENT TimeUnit ('minute' | 'hour' | 'day' | 'week')>
<!-- text------------------------------------------------------------------------------->
<!ELEMENT Name ('' | #PCDATA)>
<!ELEMENT FieldName#PCDATA>
<!ELEMENT (Path | SettingPathh) #PCDATA>
<!ELEMENT ReplicaID #PCDATA>
<!ELEMENT Title #PCDATA>
<!ELEMENT Currency #PCDATA>
<!-- logic------------------------------------------------------------------------------>
<!ELEMENT (IsBold | IsItalic | IsStrikethru | IsUnderline) ('true' | 'false')>
<!ELEMENT IsShow ('true' | 'false')>
<!-- integer (long) -------------------------------------------------------------------->
<!ELEMENT ActivityNo  #PCDATA>
<!ELEMENT MapNo| ParentMapNo #PCDATA>
<!ELEMENT Angle #PCDATA>
<!ELEMENT (Zoom | ZoomMainMap) #PCDATA>
<!ELEMENT MaximumInstances #PCDATA>
<!ELEMENT MaximumIterations #PCDATA>
<!-- numeric---------------------------------------------------------------------------->
<!ELEMENT (Size| Width | Height | Length| Offset )  #PCDATA>
<!ELEMENT (Position | LabelOffset ) (X, Y)>
```

```
<!ELEMENT (X | Y) #PCDATA>
<!ELEMENT (Value | Deviation)  #PCDATA>
```

Table 1. Selected Values of the *t*-Distribution

v	$t_{v,\,0.995}$	$t_{v,\,0.99}$	$t_{v,\,0.975}$	$t_{v,\,0.95}$	$t_{v,\,0.90}$	$t_{v,\,0.80}$	$t_{v,\,0.75}$	$t_{v,\,0.70}$	$t_{v,\,0.60}$	$t_{v,\,0.55}$
1	63.66	31.82	12.71	6.31	3.08	1.376	1.00	0.727	0.325	0.158
2	9.92	6.96	4.30	2.92	1.89	1.061	0.816	0.617	0.289	0.142
3	5.84	4.54	3.18	2.35	1.64	0.978	0.765	0.584	0.277	0.137
4	4.60	3.75	2.78	2.13	1.53	0.941	0.741	0.569	0.271	0.134
5	4.03	3.36	2.57	2.02	1.48	0.920	0.727	0.559	0.267	0.132
6	3.71	3.14	2.45	1.94	1.44	0.906	0.718	0.553	0.265	0.131
7	3.50	3.00	2.36	1.90	1.42	0.896	0.711	0.549	0.263	0.130
8	3.36	2.90	2.31	1.86	1.40	0.889	0.706	0.546	0.262	0.130
9	3.25	2.82	2.26	1.83	1.38	0.883	0.703	0.543	0.261	0.129
10	3.17	2.76	2.23	1.81	1.37	0.879	0.700	0.542	0.260	0.129
11	3.11	2.72	2.20	1.80	1.36	0.876	0.697	0.540	0.260	0.129
12	3.06	2.68	2.18	1.78	1.36	0.873	0.695	0.539	0.259	0.128
13	3.01	2.65	2.16	1.77	1.35	0.870	0.694	0.538	0.259	0.128
14	2.98	2.62	2.14	1.76	1.34	0.868	0.692	0.537	0.258	0.128
15	2.95	2.60	2.13	1.75	1.34	0.866	0.691	0.536	0.258	0.128
16	2.92	2.58	2.12	1.75	1.34	0.865	0.690	0.535	0.258	0.128
17	2.90	2.57	2.11	1.74	1.33	0.863	0.689	0.534	0.257	0.128
18	2.88	2.55	2.10	1.73	1.33	0.862	0.688	0.534	0.257	0.127
19	2.86	2.54	2.09	1.73	1.33	0.861	0.688	0.533	0.257	0.127
20	2.84	2.53	2.09	1.72	1.32	0.860	0.687	0.533	0.257	0.127
21	2.83	2.52	2.08	1.72	1.32	0.859	0.686	0.532	0.257	0.127
22	2.82	2.51	2.07	1.72	1.32	0.858	0.686	0.532	0.256	0.127
23	2.81	2.50	2.07	1.71	1.32	0.858	0.685	0.532	0.256	0.127
24	2.80	2.49	2.06	1.71	1.32	0.857	0.685	0.531	0.256	0.127
25	2.79	2.48	2.06	1.71	1.32	0.856	0.684	0.531	0.256	0.127
26	2.78	2.48	2.06	1.71	1.32	0.856	0.684	0.531	0.256	0.127
27	2.77	2.47	2.05	1.70	1.31	0.855	0.684	0.531	0.256	0.127
28	2.76	2.47	2.05	1.70	1.31	0.855	0.683	0.530	0.256	0.127
29	2.76	2.46	2.04	1.70	1.31	0.854	0.683	0.530	0.256	0.127
30	2.75	2.46	2.04	1.70	1.31	0.854	0.683	0.531	0.256	0.127
40	2.70	2.42	2.02	1.68	1.30	0.851	0.681	0.529	0.255	0.126
60	2.66	2.39	2.00	1.67	1.30	0.848	0.679	0.527	0.254	0.126
120	2.62	2.36	1.98	1.66	1.29	0.845	0.677	0.526	0.254	0.126
∞	2.58	2.33	1.96	1.645	1.28	0.842	0.674	0.524	0.253	0.126
v	$t_{v,\,0.995}$	$t_{v,\,0.99}$	$t_{v,\,0.975}$	$t_{v,\,0.95}$	$t_{v,\,0.90}$	$t_{v,\,0.80}$	$t_{v,\,0.75}$	$t_{v,\,0.70}$	$t_{v,\,0.60}$	$t_{v,\,0.55}$

v: degrees of freedom

Source: [Gottfried, 1984, pp. 168-169]

Table 2. Selected Values of the *N*(0, 1) Distribution

Z	0	1	2	3	4	5	6	7	8	9
0.0	0.0000	0.0040	0.0080	0.0120	0.0160	0.0199	0.0239	0.0279	0.0319	0.0359
0.1	0.0398	0.0438	0.0478	0.0517	0.0557	0.0596	0.0636	0.0675	0.0714	0.0754
0.2	0.0793	0.0832	0.0871	0.0910	0.0948	0.0987	0.1026	0.1064	0.1103	0.1141
0.3	0.1179	0.1217	0.1255	0.1293	0.1331	0.1368	0.1406	0.1443	0.1480	0.1517
0.4	0.1554	0.1591	0.1628	0.1664	0.1700	0.1736	0.1772	0.1808	0.1844	0.1879
0.5	0.1915	0.1950	0.1985	0.2019	0.2054	0.2088	0.2123	0.2157	0.2190	0.2224
0.6	0.2258	0.2291	0.2324	0.2357	0.2389	0.2422	0.2454	0.2486	0.2518	0.2549
0.7	0.2580	0.2612	0.2642	0.2673	0.2704	0.2734	0.2764	0.2794	0.2823	0.2852
0.8	0.2881	0.2910	0.2939	0.2967	0.2996	0.3023	0.3051	0.3078	0.3106	0.3133
0.9	0.3159	0.3186	0.3212	0.3238	0.3264	0.3289	0.3315	0.3340	0.3365	0.3389
1.0	0.3413	0.3438	0.3461	0.3485	0.3508	0.3531	0.3554	0.3577	0.3599	0.3621
1.1	0.3643	0.3665	0.3686	0.3708	0.3729	0.3749	0.3770	0.3790	0.3810	0.3830
1.2	0.3849	0.3869	0.3888	0.3907	0.3925	0.3944	0.3962	0.3980	0.3997	0.4015
1.3	0.4032	0.4049	0.4066	0.4082	0.4099	0.4115	0.4131	0.4147	0.4162	0.4177
1.4	0.4192	0.4207	0.4222	0.4236	0.4251	0.4265	0.4279	0.4292	0.4306	0.4319
1.5	0.4332	0.4345	0.4357	0.4370	0.4382	0.4394	0.4406	0.4418	0.4429	0.4441
1.6	0.4452	0.4463	0.4474	0.4484	0.4495	0.4505	0.4515	0.4525	0.4535	0.4545
1.7	0.4554	0.4564	0.4573	0.4582	0.4591	0.4599	0.4608	0.4616	0.4625	0.4633
1.8	0.4641	0.4649	0.4656	0.4664	0.4671	0.4678	0.4686	0.4693	0.4699	0.4706
1.9	0.4713	0.4719	0.4726	0.4732	0.4738	0.4744	0.4750	0.4756	0.4761	0.4767
2.0	0.4772	0.4778	0.4783	0.4788	0.4793	0.4798	0.4803	0.4808	0.4812	0.4817
2.1	0.4821	0.4826	0.4830	0.4834	0.4838	0.4842	0.4846	0.4850	0.4854	0.4857
2.2	0.4861	0.4864	0.4868	0.4871	0.4875	0.4878	0.4881	0.4884	0.4887	0.4890
2.3	0.4893	0.4896	0.4898	0.4901	0.4904	0.4906	0.4909	0.4911	0.4913	0.4916
2.4	0.4918	0.4920	0.4922	0.4925	0.4927	0.4929	0.4931	0.4932	0.4934	0.4936
2.5	0.4938	0.4940	0.4941	0.4943	0.4945	0.4946	0.4948	0.4949	0.4951	0.4952
2.6	0.4953	0.4955	0.4956	0.4957	0.4959	0.4960	0.4961	0.4962	0.4963	0.4964
2.7	0.4965	0.4966	0.4967	0.4968	0.4969	0.4970	0.4971	0.4972	0.4973	0.4974
2.8	0.4974	0.4975	0.4976	0.4977	0.4977	0.4978	0.4979	0.4979	0.4980	0.4981
2.9	0.4981	0.4982	0.4982	0.4983	0.4984	0.4984	0.4985	0.4985	0.4986	0.4986
3.0	0.4987	0.4987	0.4987	0.4988	0.4988	0.4989	0.4989	0.4989	0.4990	0.4990
3.1	0.4990	0.4991	0.4991	0.4991	0.4992	0.4992	0.4992	0.4992	0.4993	0.4993
3.2	0.4993	0.4993	0.4994	0.4994	0.4994	0.4994	0.4994	0.4995	0.4995	0.4995
3.3	0.4995	0.4995	0.4995	0.4996	0.4996	0.4996	0.4996	0.4996	0.4996	0.4997
3.4	0.4997	0.4997	0.4997	0.4997	0.4997	0.4997	0.4997	0.4997	0.4997	0.4998
3.5	0.4998	0.4998	0.4998	0.4998	0.4998	0.4998	0.4998	0.4998	0.4998	0.4998
3.6	0.4998	0.4998	0.4999	0.4999	0.4999	0.4999	0.4999	0.4999	0.4999	0.4999
3.7	0.4999	0.4999	0.4999	0.4999	0.4999	0.4999	0.4999	0.4999	0.4999	0.4999
3.8	0.4999	0.4999	0.4999	0.4999	0.4999	0.4999	0.4999	0.4999	0.4999	0.4999
3.9	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000
Z	0	1	2	3	4	5	6	7	8	9

Source: [Gottfried, 1984, pp. 172-173]

Table 3. Selected Values of the χ^2 Distribution

v	$\chi^2_{v,\,0.99}$	$\chi^2_{v,\,0.95}$	$\chi^2_{v,\,0.75}$	$\chi^2_{v,\,0.50}$	$\chi^2_{v,\,0.25}$	$\chi^2_{v,\,0.05}$	$\chi^2_{v,\,0.01}$
1	0.00016	0.00393	0.1015	0.4549	1.322	3.841	6.635
2	0.00201	0.1026	0.5753	1.386	2.773	5.991	9.210
3	0.1148	0.3518	1.213	2.366	4.108	7.815	11.34
4	0.2971	0.7107	1.923	3.357	5.385	9.488	13.28
5	0.5543	1.1455	2.675	4.351	6.626	11.07	15.09
6	0.8720	1.635	3.455	5.348	7.841	12.59	16.81
7	1.239	2.167	4.255	6.346	9.037	14.07	18.48
8	1.646	2.733	5.071	7.344	10.22	15.51	20.09
9	2.088	3.325	5.899	8.343	11.39	16.92	21.67
10	2.558	3.940	6.737	9.342	12.55	18.31	23.21
11	3.053	4.575	7.584	10.34	13.70	19.68	24.73
12	3.571	5.226	8.438	11.34	14.84	21.03	26.22
15	5.229	7.261	11.04	14.34	18.25	25.00	30.58
20	8.260	10.85	15.45	19.34	23.83	31.41	37.57
30	14.95	18.49	24.48	29.34	34.80	43.77	50.89
50	29.71	34.76	42.94	49.33	56.33	67.50	76.15

v: degrees of freedom

Source: [Gottfried, 1984, p. 36]

Table 4. Selected Values of the *F*-Distribution for α = 0.05

$F_{m,n,\alpha}$ m / n	1	2	3	5	8	12	15	20	30	60	∞
1	161.4	199.5	215.7	230.2	238.9	243.9	245.9	248.0	250.1	252.2	254.3
2	18.51	19.00	19.16	19.30	19.37	19.41	19.43	19.45	19.46	19.48	19.50
3	10.13	9.55	9.28	9.01	8.85	8.74	8.70	8.66	8.62	8.57	8.53
4	7.71	6.94	6.59	6.26	6.04	5.91	5.86	5.80	5.75	5.69	5.63
5	6.61	5.79	5.41	5.05	4.82	4.68	4.62	4.56	4.50	4.43	4.36
6	5.99	5.14	4.76	4.39	4.15	4.00	3.94	3.87	3.81	3.74	3.67
7	5.59	4.74	4.35	3.97	3.73	3.57	3.51	3.44	3.38	3.30	3.23
8	5.32	4.46	4.07	3.69	3.44	3.28	3.22	3.15	3.08	3.01	2.93
9	5.12	4.26	3.86	3.48	3.23	3.07	3.01	2.94	2.86	2.79	2.71
10	4.96	4.10	3.71	3.33	3.07	2.91	2.85	2.77	2.70	2.62	2.54
11	4.84	3.98	3.59	3.20	2.95	2.79	2.72	2.65	2.57	2.49	2.40
12	4.75	3.89	3.49	3.11	2.85	2.69	2.62	2.54	2.47	2.38	2.30
13	4.67	3.81	3.41	3.03	2.77	2.60	2.55	2.46	2.38	2.30	2.21
14	4.60	3.74	3.34	2.96	2.70	2.53	2.46	2.39	2.31	2.22	2.13
15	4.54	3.68	3.29	2.90	2.64	2.48	2.40	2.33	2.25	2.16	2.07
16	4.49	3.63	3.24	2.85	2.59	2.42	2.35	2.28	2.19	2.11	2.01
17	4.45	3.59	3.20	2.81	2.55	2.38	2.31	2.23	2.15	2.06	1.96
18	4.41	3.55	3.16	2.77	2.51	2.34	2.27	2.19	2.11	2.02	1.92
19	4.38	3.52	3.13	2.74	2.48	2.31	2.23	2.16	2.07	1.98	1.88
20	4.35	3.49	3.10	2.71	2.45	2.28	2.20	2.12	2.04	1.95	1.85
21	4.32	3.47	3.07	2.68	2.42	2.25	2.18	2.10	2.01	1.92	1.82
22	4.30	3.44	3.05	2.66	2.40	2.23	2.15	2.07	1.98	1.89	1.79
23	4.28	3.42	3.03	2.64	2.37	2.20	2.13	2.05	1.96	1.86	1.76
24	4.26	3.40	3.01	2.62	2.36	2.18	2.11	2.03	1.94	1.84	1.73
25	4.24	3.39	2.99	2.60	2.34	2.16	2.09	2.01	1.92	1.82	1.71
26	4.23	3.37	2.98	2.59	2.32	2.15	2.07	1.99	1.90	1.80	1.69
27	4.21	3.35	2.96	2.57	2.31	2.13	2.06	1.97	1.88	1.79	1.67
28	4.20	3.34	2.95	2.56	2.29	2.12	2.04	1.96	1.87	1.77	1.65
29	4.18	3.33	2.93	2.55	2.28	2.10	2.03	1.94	1.85	1.75	1.64
30	4.17	3.32	2.92	2.53	2.27	2.09	2.01	1.93	1.84	1.74	1.62
40	4.08	3.23	2.84	2.45	2.18	2.00	1.92	1.84	1.74	1.64	1.51
60	4.00	3.15	2.76	2.37	2.10	1.92	1.84	1.75	1.65	1.53	1.39
120	3.92	3.07	2.68	2.29	2.02	1.83	1.75	1.66	1.55	1.43	1.25
∞	3.84	3.00	2.60	2.21	1.94	1.75	1.67	1.57	1.46	1.32	1.00
n / $F_{m,n,\alpha}$ m	1	2	3	5	8	12	15	20	30	60	∞

m: degrees of freedom in the numerator
n: degrees of freedom in the denominator
Source: [Maisel/Gnugnoli, 1972, p. 400]

INDICES

Symbolic Definitions

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	63
$T^S(j)$:	the set of outgoing links of activity j	15
$U(a, b)$:	The uniform distribution or a rectangular distribution with values in range [a, b]	110
$U[a, b]$:	The uniform distribution or a rectangular distribution with integer values in range [a, b]	148

Concept Definitions

Assumptions

Algorithms

Figures

Examples

PM RELEASE STREAM

In 1994, Marcus Ott developed the modeling tool "Workflow Modeling Editor" (WOMED) for the GroupFlow system running on the Notes platform (see [Nastansky/Hilpert, 1996]). Microsoft Visual Basic V.3.0 was used as the development tool. WOMED is a preliminary process-modeling prototype with a graphical user interface for intuitively and straightforwardly designing a flexible process definition. Routing options and team workflow participants can be utilized in the process definition (see [Ott, 1994]). Using a clustering mechanism, the "WOMED workflow modeling editor seamlessly supports simultaneous top-down and bottom-up planning cycles for business processes." [Nastansky/Hilpert, 1994]

In August 1994, the author began integrating the simulator into the modeling tool, which was at that time called GroupFlow Modeler 1.2 and was being further developed by Ralf Heindörfer. Since 1996, the author has been in charge of the professional development of the whole modeling tool. The modeler has been renamed to ProcessModeler and is developed with Visual Basic V.4.0 for the 32-bit Windows system. The definitions and algorithms discussed in this work are implemented in the modeler step by step.

- *GroupFlow Modeler 1.2*, 1994-1996: verify the network (start activity, disconnected part of the activity network, infinite cycles, etc.) of a process definition; animate and simulate process instances; design the simulation database.
- *ProcessModeler 2.0* (builds 100-168), 1996-1997: determine copy flags for joining; improve the source codes by using the new features provided by VB V.4.0; open multiple process definitions; undo changes; print the network in multiple papers; define multiple activities simultaneously; go to a certain activity on the network.
- *ProcessModeler 2a* (builds 164-202), 1997-1998: determine workflow control data; assign and simulate the join priorities for releasing deadlocks; simulate costs and material resources; specify and simulate the team of the fixed participants for an activity; copy/paste part or all of the network; improve performance for loading a large organization's data and for saving a large process definition.
- *ProcessModeler 3.0* (builds 218-484), 1998-2000: integrate the Analyzer for evaluating the running process instances of a process definition; improve algorithms for determining workflow control data; define and simulate multiple start activities; simulate variables (also activity-dependence and multiple-values) defined in routing conditions; specify and simulate escalation data; separate simulation settings of the organization model; utilize and simulate the Notes Address Book; inform of released deadlocks during animation; specify and simulate workflow participants of the previous activity or the supervisor of the

process definition; simulate the workflow initiator; make sound during a simulation run; present distribution functions for selection; continue a simulation run; display the results of integrity verification; verify outgoing links of a Notes Agent activity; define the stand-in of workflow participants to an activity; present potential deadlock situations; copy part or all of a process map for pasting it to other graphic applications; manage versions of a process definition; define and select a layout for the process map; present a flat view of the network of a process definition with clusters; include keyboard actions in addition to mouse operations; organize icons specified to activities hierarchically; implement the explorer; add zoom feature; print a table in accordance with the paper size; export a table to a CSV file; run a test version of a process definition further after making changes; define enterprise links and create Notes real-time activities for exchanging data with relational databases; improve performance for verifying large process definitions, for accessing databases and for drawing the networks of large process definition with many clusters.

- *ProcessModeler 5.0* (builds 1000-1048), 2000-2001: save animation settings and reports; improve performance for detecting deadlocks; develop stand-alone PM (all data are kept in XML files instead of Notes databases); define categories for a process definition; check spelling of texts on the process map; flip part or all of a process map diagonally, horizontally or vertically; keep definition dialogue boxes on the top of the screen and switch between them according to the selection on the process map; preview a page when printing process maps or a table; read only filtered process instances in the Analyzer.
- *ProcessModeler 5.0-next* (builds 1060-1076), 2001: integrate the ProcessViewer, which presents the process maps just like PM and animates the routing of a process instance from activity to activity and from person to person, in a Notes application database integrated with the Espresso workflow engine; save process definition data according to the requirements of the ProcessViewer developed with JAVA; specify never-join activities; overview the definitions of all activities as well as links in table forms; keep user-adjusted column widths of a table; specify main document for splitting (like that for joining); specify initial page overlap size for printing; improve the display of label and pop-up messages of an activity and a link.

DECLARATION

I hereby declare that this dissertation is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgement has been made in the text.

Hong Zhang
Paderborn, August 31. 2001