Dynamic Change Within Workflow Systems

Clarence Ellis,
Karim Keddara
Dept. Computer Science,
University of Colorado,
Boulder, Co. 80309-0430,
USA.

Grzegorz Rozenberg
Dept. Computer Science,
University of Leiden,
Niels Bohrweg 1,
2300 RA Leiden,
The Netherlands.

Abstract

Dynamic change is a large and pervasive unsolved problem which surfaces within office systems as well as within software engineering, manufacturing, and numerous other domains. Procedural changes, performed in an ad hoc manner, can cause inefficiencies, inconsistencies, and catastrophic breakdowns within offices. This paper is concerned with dynamic change to procedures in the context of workflow systems. How can we make workflow systems more flexible and open? We believe that part of the answer lies in the study and solution of the dynamic change problem. In this paper, we use a Petri net formalism to analyze structural change within office procedures. As an example, we define a class of change called "synthetic cut-over change", and apply our formalism to prove that this class maintains correctness when downsizing occurs.

Keywords: Dynamic Change, Office Procedures, Workflow Systems, Petri Nets, Organizational Evolution

1 Introduction

Contemporary organizations employ a vast array of computing technology to support their information processing needs. There are many successful computing tools designed as personal information aids (word processors, spreadsheets, etc.) but fewer tools designed for collaborating groups of people (groupware). Many groupware products have recently been introduced to the market [1]. A few of these products capture knowledge of the organizational activity that they are assisting, but the vast majority do not.

COOCS 95 Milpitas CA USA © 1995 ACM 0-89791-706-5/95/08..$3.50

For example, a group document editor knows nothing about the organizational purpose of the document being edited. Organizationally aware groupware can potentially lead to significantly more powerful and useful systems. One class of organizationally aware groupware is workflow.

Workflow systems are designed to assist groups of people in carrying out work procedures, and contain organizational knowledge of where work flows in the default case. Workflow is defined as "systems that help organizations to specify, execute, monitor, and coordinate the flow of work items within a distributed office environment" [5]. The system contains two basic components: the first component is the workflow model (or "specification module"), which enables administrators and analysts to define procedure and activities, analyze and simulate them, and assign them to people. This component can model goals, control structures, data structures, organizational structures, conversation structures, etc. Most workflow models capture (at least) procedures and the steps which make up the procedures and the precedence ordering between steps. In this document, we model procedures with a sprecial kind of Petri nets, called in the sequel flow nets, we model the steps within the procedure (called activities) as transitions, and precedence (i.e. the "preceeds" ordering relation between activities) as arcs in the flow net. We assume that the reader has basic knowledge of Petri nets (see Figure [1] for an example.) We ignore other important workflow components such as roles, agents, repositories, resources, etc. It turns out that our dynamic change analysis is applicable to other components, and that dynamic changes to other components are frequently less complex than activity/precedence changes.

The second component is the workflow execution module (the workflow enactment system) consisting of the execution interface seen by end users and the execution environment which assists in coordinating and

performing the procedures and activities. It enables the units of work to flow from one user's workstation to another as the steps of a procedure are completed.

How do the first and second components relate? We believe that the specification and execution modules need to be tightly interwoven. For example, it should be possible to change the workflow model of a procedure, and thereby dynamically change how the steps of the procedure are being executed. Our belief is based upon the observation that change is a way of life in most organizational and personal settings. Those organizations in the modern business world which refuse to change are headed toward rapid obsolescence because they cannot compete. Organizations must frequently make structural changes such as:

- adding a new employee,
- adjusting for a new tax law,
- filling in for a manager on vacation.

Changes often dictate other concomitant changes, so it is often necessary to do a set of changes as a unit. Dynamic change problems have been documented in the workflow litterature [9]. This can get very complex and error prone. In practice many organizations find it necessary to suspend or abort the work in progress in order to avoid undesirable side effects of complex changes. This is an inefficient, and ineffective change process because many organizations find it very unproductive, and sometimes impossible to shut down all activities in order to make changes. From pharmaceutical factories to software engineering houses, this is a nagging problem-the bigger the organization, the more complex are the procedures, and the more painful the change processes. Today, organizations usually do not solve this problem, they cope, evade, or "muddle through." This paper addresses this dynamic change problem, and verifies the correctness of one class of dynamic change.

We are concerned with dynamic structural change. Structural means that we are concerned with changes to the structure of procedures; we are not concerned here with changes to the value of an application data variable. Dynamic means that we are required to make the change "on the fly" in the midst of continuous execution of the changing procedure. We restrict our consideration to structural changes concerning the steps of a procedure (called activities) and their precedence. Examples include changes such as deletion of an activity, addition of a precedence relation between two activities, and parallelization of two activities that previously were constrained to execute serially. A very simple example of dynamic structural change within an office procedure is the following. An organization which traditionally does order processing performs the shipping step and the billing steps at the same time, makes a dynamic change to its procedure by performing the shipping step after the billing step Although the procedure "looks safe" before the change, and "looks safe" to all orders processed after the change, there are problems that could potentially surface during the change. For example, since orders that are in progress during the change are not flushed, some of these orders which went through the shipping step but not through the billing set, will never perform the billing step at all, so some customers will not be billed. This example, used throughout this paper, is explained in more detail later.

2 Related Work and CSCW Context

There has been considerable work and publication related to workflow systems, models, and studies. Historically, these systems grew out of the office information systems of the 1970s; Workflow systems have been categorized in the literature based upon the models from which they are derived - see, for example, the article by Bair in the 1993 Groupware Conference Proceedings [3]. Although studies and models of work practices and work procedures have spanned the gamut from very informal to very formal, the vast majority of workflow products are based upon relatively rigid and formal procedural models. Notable exceptions include the ActionWorkflow product [9] based upon a speech act conversation model and the Polymer research prototype [7] based upon a goal based planning.

Informal modeling and ethnographic studies have been reported by Suchman, Wynn, and other cultural anthropologists. Considerable effort has been put into workflow studies within the Information Systems field and the Organizational Design field within business schools[10]. Several procedural office models have emerged from concepts of discrete mathematics, and the software engineering community including graph based models such as Petri Nets and matrix algebra models. Articles on a variety of workflow systems and models can be found in various proceedings of past ACM SIGOIS conferences, and the annual groupware conferences. Office models are reviewed and contrasted in several articles[4]. None of these models address the problem of dynamic structural change.

The problem of dynamic structural change has surfaced in numerous domains including CIM (Computer Integrated Manufacturing)[16] and Software Engineering[12]. Mathematical models that have arisen from these domains include extended flowchart
state machine notations, project management models, and process programming models. S.K. Chang notes the utility of transformation and verification of office procedures, but does not address the dynamic change problem [6]. One problem with many of these mathematical models is that they are basically designed to

analyze static structures. Thus, although a finite automaton or Petri net can analyze change from state to state, reachability, and deadlock, it has no mechanism to analyze the addition of new states nor the alteration of state structure. This is especially true if these changes are not a priori known.

Although there have been workflow success stories, it is generally acknowledged that workflow has not lived up to its expectation [2]. Workflow seems to fail more often than succeed. Traditional workflow systems (and office information systems before them) have been criticized in the literature as "automating a fiction" in the office because of their tendency to inflexibly prescribe temporal activity sequencing, and to narrowly dictate and restrict, rather than to broadly assist in the roles people play. People in offices typically engage in lots of problem solving, informal communication, and exception handling. In order to "get the work done" it may be necessary to creatively augment or circumvent standard office procedures. The mechanisms to help people do their necessary problem solving and exception handling are typically lacking in today's workflow systems. Office work has been better characterized as "situated action" and "articulation work" [15], than its older description, derived from scientific management literature, as detailed procedure execution.

One response to this criticism has been the rejection of workflow and formal models, and the emphasis on "groupware tools" which have no knowledge of the organizational context, e.g. group editors. We believe that there is great potential for groupware which is goal cognizant, and organizationally aware, but we agree that significant research is needed to realize this potential. We also feel that progress in the cscw arena requires multiple disciplines, tools, and approaches. In this spirit, careful, cognizant formal modeling of human endeavors can potentially provide valuable insight. Another response has arisen from the business community, saying that there are significant examples of successful workflow, so we must continue to sell workflow and to incrementally improve it. We believe there can be significant learning by doing this. We hope that it is coupled with a paradigm shift away from the emphasis on prescriptive procedure enforcement.

The authors are associated with an ongoing research group, the Collaboration Technology Research Group (CTRG), at the University of Colorado and at Southern University, which is actively addressing these issues within our "Next Generation Workflow" research project. Within CTRG, our response has been research work to redirect the emphasis of work flow to dynamic goal based systems [17]. Members of CTRG have conducted numerous office studies, and built workflow systems. A frequent reaction to the description and model produced by the study is "This is an interesting view of our office, but we don't do our work like this anymore - we've changed. " A frequent reaction to the installation of workflow systems is "Nice technology, but it doesn't allow us the flexibility to handle the many exceptions, and to really get our work done expeditiously." Dynamic change can help to address these statements. We have found that in many environments, workflow can be very helpful if it is dynamic, flexible, changeable, knowledgable, and open. Our ongoing CTRG work strives to avoid the pitfalls articulated by Robinson and Bannon [14] by:

- not imposing an order on events or people, but optionally displaying what has been done (and by whom) in the past,
- not precluding people from, at any time, reworking the model, but encouraging and assisting in evolutionary change and exception handling,
- not insisting that the model be determinant and consistent, but allowing multiple interpretations of multiple realities.

We argue that workflow systems do not need to be dictators; they can be friendly assistants that help you reason about your work. They are available when and if you want them. This paper describes one important component of our CTRG research effort. One type of reasoning help is to reason about procedural change (both temporary and permanent) within structured work. To perform this type of reasoning, it is useful to have formal definitions and apply mathematical analyses.

Models of workflow can be quite useful and informative planning tools without being used as an execution component Presentation of multiple views of how an organization is perceived to work (or how it has done procedures in the past,) as well as other information presented by the model, can be very useful to workers without any automation. Different degrees of proceduralization, and different types of computer augmentation are appropriate for different types of organizations. Thus, the work in this paper is independent of any execution component of any particular workflow system; this is particularly appropriate if the organization performs primarily unstructured activity.

The rest of the paper is organized as follows: In section 3 we introduce the running example which will be used throughout the paper. In section 4 we establish our mathematical notations. In section 5, the notion of flow net is introduced as a model of workflow procedures, we also recall some well-known notions from the Petri net theory. Next, the dynamic change within workflow procedures as well as the synthetic cut-over change are modeled in section 6 in terms of net replacemment. Followed, in section 7, by the introduction of the notion of correctness of dynamic change. Finally, our main results are stated in section 8.

3 A Dynamic Change Example

This paper presents a formal definition of dynamic change, and a mathematical approach to its analysis. We stress that this analysis is to be used interactively and synergistically, with end users mediating the social and organizational aspects of the changes [10]. Some changes are easy, some are difficult. It is typically easy to make an isolated change to the value of a variable in a database - this is considered "normal". Likewise, change of policy in many organizations is considered "normal," e.g. 'Our future policy will reimburse our employees 30c per mile, rather than the previous 20c per mile.' These types of changes tend to be easier to implement than structural change. If we consider a procedure as one type of structure within an organization, then change to that procedure is structural change. One company, when audited, found that they did not have sufficient separation of functional control within their procedures, and was required to make severe structural change that transcended the boundaries of many procedures. This is the type of complex change that our analysis can greatly assist.

This type of dynamic change can at times encounter "dynamic bugs" which would not appear within more static change. As an example of the type of "dynamic bug" problem that we are addressing,

Example 3.1 *consider an office procedure for order processing within a typical electronics company. When a customer requests by mail, or in person, an electronic part, this is the beginning of a job (also called a work case.) A form is filled out by the order administrator; the job is sent to credit check, and then to inventory check. After the evaluation, either a rejection letter is sent to the customer, or the order is approved and then sent to shipping and billing. The shipping department will actually cause the part to be sent to the customer; the billing department will see that the customer is sent a bill, and that it is paid. This procedure is shown in Figure [1].*

Suppose that the organization decides to initiate the credit checking and the inventory checking steps at the same time for speedier processing (see Figure[3]). This is an example of structural change because the structure of the procedure is changing. An even simpler structural change that we will analyze is to move the billing step to take place before the shipping step (see Figure[6])- there could be many reasons for wanting to do this. One way to do this change could be to delay and not process any new customer requests until after the change, and simultaneously, wait until all ongoing jobs are completed before making the change. This means that no jobs are in progress when the change is made. This strategy, called flushing the system, is safe, but quite costly - it might take years for the current jobs (perhaps thousands) to all reach completion, and this may delay thousands of new customers for an unacceptably long time. Another unpleasant strategy is to abort all jobs in progress. Another is to have the old version and the new version of the procedure simultaneously available. There are variations of these strategies that are used, which have more or less safety. In this paper, we are concerned with making structural changes safely without flushing the system. This is the definition of dynamic change. In many situations, much can be gained if we can understand, and safely perform dynamic structural change. Typically, the more quickly we can convert all jobs to this change, the better.

A dynamic change problem occurs in our example if a job has been processed by shipping at the time of the change but not by the billing. This job is then sent to archive according to the instructions of the new procedure Thus a customer will not be billed for the part that he receives. This situation is depicted graphically in Figure [6]. If there are a large number of jobs being in the same situation at the time of change, then a large number of customers will not be billed. This is a very simple example of a "dynamic bug;" many of these bugs are much more difficult to detect and can have strange and insidious effects.

Our approach to analyzing change is mathematically detailed in later sections of this document, and can be informally summarized as follows. Given a specific procedural change, we define its change region as the part of the net containing all the activities directly affected by the change. The old region is the change region prior to the change, and the new region is the change region after the change. These notions of change regions will be discussed later. We think of the change as replacing the old region by the new region within the specification of the procedure (see Figures [1,2,3]). The jobs evolving outside the change region are not affected by the change. The jobs inside the old region are "transferred" to the new region. This transfert can result in the creation of new jobs or the destruction of old jobs.

After a change takes place, the work resumes its progression in a new environement as described by the new procedure. The change is said to be correct if the resumption is intended to finish the in-progress work according to either the old or the new procedure. Clearly, this correctness criterion allows us to capture the dynamic bugs described earlier. In some cases, the new change region is such that it contains both the old and the new region (see Figures [7,8,9]). This class of changes, referred to as synthetic cut-over change, is in some cases safer than the immediate change. For instance when downsizing occurs (i.e. the new region can do less than the old region), we can prove that the synthetic cut-over change is correct.

4 Preliminaries

In this section we recall some basic mathematical notions and we establish our notation and terminology.
The set of integers is denoted $\mathbf{N}$ and $\mathbf{N}^+$ denotes the set of positive integers. For a finite alphabet Σ, Σ^* denotes the set of all finite words over Σ and λ denotes the empty word. The concatenation of two words w and w' is denoted ww'. For $w_1, w_2 \in \Sigma^*$, the shuffle of w_1 and w_2, denoted $w_1 \| w_2$ is defined inductively as follows:

$$a\|\lambda = \lambda \|a = a \;\&\; ax_1\|bx_2 = a(x_1\|bx_2) \cup b(ax_1\|x_2)$$

for $a, b \in \Sigma$ and $x_1, x_2 \in \Sigma^*$. As usual this operator is extended to languages; $L_1 \| L_2 = \{w_1 \| w_2 \mid w_1 \in L_1, w_2 \in L_2\}$.

5 Workflow Procedure Modeling

The Petri Net model [13] is a simple, yet rigorous mathematical formalism, which has been used to model systems which exhibit concurrency, communication and choice between different courses of actions. They have a nice graphical representation which offers a very clear impression of the concurrent and nondeterministic aspects of the systems they model. A workflow procedure is modeled by a flow net. It is a Petri Net with two distinguished places; namely the input place and the output place. The activities of the procedure are modeled by transitions, each of which has a name, at least one input place and at least one output place. Formally,

Definition 5.1 *Let Σ be a finite alphabet of activity names. A* **flow net** *over Σ (net for short) is a system $M = (S, T, F, Lab; s_{in}, s_{out})$ which consists of:*
• *disjoint, finite and non empty sets S of* **places** *and T of* **transitions**.
• *$F \subseteq (S \times T) \bigcup (T \times S)$ the* **flow relation** *which satisfies the following properties:*

$$\forall x \in S, {}^\bullet x \bigcup x^\bullet \neq \emptyset.$$

$$\forall t \in T, {}^\bullet t \neq \emptyset \; and \; t^\bullet \neq \emptyset.$$

• *$Lab : T \longrightarrow \Sigma$ the* **transition labeling** *function.*
• *$s_{in} \in S$ the* **input place** *of M, and $s_{out} \in S$ the* **output** *place of M which are such that: ${}^\bullet s_{in} = \emptyset$ and $s_{out}^\bullet = \emptyset$. Moreover, the set $\{s_{in}, s_{out}\}$ is called the* **interface** *of M.*

The notion of marking and marked nets are defined as usual. A function $m : S \longrightarrow \mathbf{N}$ is called a marking. In particular, $\emptyset$ denotes the empty marking, and if $i \in \mathbf{N}$, then $\underline{i}$ (resp. $\overline{i}$) is the unique **initial** (resp. **terminal**) marking which consists of i tokens in the input (resp. output) place and zero tokens elsewhere. $\underline{Mark}(M)$ denotes the class of all markings of M. A marked net $\mathcal{M} = (M; m)$ consists of a net M and a marking m of M.
The dynamic component of a net evolves around the well-known notion of transition firing, and firing sequences. Formally,
• Let M be a net, and m be a marking of M. A transition t of M is enabled under m, written $m[t\rangle$ iff $\forall s \in {}^\bullet t, m(s) \neq 0$. In this case the firing of t, denoted $m[t\rangle m'$, is said to lead to the marking m' where:

$$m'(s) = m(s) - 1 \text{ if } s \in {}^\bullet t - t^\bullet,$$
$$m'(s) = m(s) + 1 \text{ if } s \in t^\bullet - {}^\bullet t,$$
$$m'(s) = m(s) \text{ otherwise.}$$

• Let M be a net, let m and m' be markings of M, and let $w \in T^*$. Then w is an m-firing sequence leading to m' iff either $w = \lambda$ and $m = m'$ or $w = w't$ with $t \in T$, $m[w'\rangle m''$ and $m''[t\rangle m'$ for some marking m'' of M and some $w' \in T^*$. In this case m' is said to be **reachable** from m. $\underline{Fire}(M, m, m')$ denotes the language of all m-firing sequences leading to m' and $\underline{Reach}(M, m)$ denotes the class of all markings which are reachable from m. This notion is lifted to the level of activity names by considering the sequence of names that compose a firing sequence. Thus, if w is an m-firing sequence leading to m', then $u = Lab^*(w)$ is a labeled m-firing sequence leading to m'. The language of all labeled m-firing sequences leading to m' is denoted $\underline{Lang}(M, m, m')$.

Example 5.1 *In the graphical representation of a marked net , a transition t labeled u is drawn as a thick line segment with the label u next to it, a place is drawn as a circle , the flow relation as a set of edges and a token is drawn as a black dot next to the place where it resides. The input and output places will be drawn as grey-colored circles, their distinction should be clear from the picture. Figure[1] depicts the office procedure for order processing which is in progress. At this stage the credit check has been completed and the inventory-check is to be initiated next. The activity names are (hopefully) clear from the context.*

An execution of a net modeling a workflow procedure starts in an initial marking, say $\underline{i}$ where $i \in \mathbf{N}^+$, and ends when one of the terminal markings, say $\overline{j}$ where $j \in \mathbf{N}^+$, is reached. Note here that an execution may take a "bad path" (e.g. deadlocks or diverges), meaning that it can reach a marking m from which it cannot reach a terminal marking. Formally,

Definition 5.2 *Let M be a net, let $m, m' \in \underline{Mark}(M)$ and let $w \in \underline{Fire}(M, m, m')$. if $m = \underline{i}$ for some $i \in \mathbf{N}^+$ and $m' = \overline{j}$ for some $j \in \mathbf{N}^+$, then u is called an* **execution sequence** *which* **consumes** *i tokens and* **produces** *j tokens.*

The next definition introduces a special kind of nets called in the sequel transactions. These are nets which

from the token input-output stand point behave like a transition which has a single input place and a single output place. In other words, each time a transaction consumes i tokens, it will produce i tokens. Furthermore, reaching a terminal marking is always guaranteed. Formally,

Definition 5.3 *A net M is a* **transaction** *iff for every $i \in \mathrm{N}^+$ the following conditions are satisfied:*
- $\bar{i} \in \underline{Reach}(M, i)$.
- $\forall m \in \underline{Reach}(M, i), \bar{i} \in \underline{Reach}(M, m)$.

In the case of a transaction, an **elementary** execution is a firing sequence which consumes 1 token (and hence produces 1 token). In some cases, many executions may be initiated at the same time. The resulting sequence is referred to as **compound** execution. Note here, that combining elementary executions results in a compound execution, but the converse does not in general hold. For instance some special measures (i.e. execution sequences) may be triggered if the load of the system reaches a certain level, and which would not be otherwise. The property of **decomposition**, introduced next, deals with this issue. Formally,

Definition 5.4 *A transaction M is* **decomposable** *iff for every $i \in \mathrm{N}^+$*

$$\underline{Fire}(N, i, i) = \underbrace{\underline{Fire}(N, 1, 1) \| \cdots \| \underline{Fire}(N, 1, 1)}_{i}.$$

Finally, we define a particular operation on marked nets which will be used later. Let $\mathcal{M}$ and $\mathcal{M}'$ be marked nets with identical interface-markings. The **fusion** of $\mathcal{M}$ and $\mathcal{M}'$, is the marked net denoted $\underline{fuse}(\mathcal{M}, \mathcal{M}')$, which is obtained by:
- removing all but the interface tokens from $\mathcal{M}'$,
- removing the tokens from the interface of $\mathcal{M}$, and
- merging the output places of both nets.

The interface of the resulting net is the interface of $\mathcal{M}'$.

Example 5.2 *The net called the new region depicted in Figure[8] is the fusion of the nets depicted in Figure[5].*

6 Dynamic Change Modeling

The change that a workflow procedure M undergoes is said to be **dynamic**. Dynamic entails that the change is made in the midst of execution (i.e. while some tokens are in progress). In terms of our net-based model, the change is viewed as the replacement of a marked subnet $\mathcal{N}_1 = (N_1; m_1)$ by a marked subnet $\mathcal{N}_2 = (N_2; m_2)$ in a marked net $\mathcal{M} = (M; m)$ which results in a marked net $\mathcal{M}' = (M'; m')$. Here, N_2 is the new version of N_1, m_1 is the token distribution in N_1 prior to the change, m_2 is the token distribution in N_2. $\mathcal{N}_1$ is referred to as the **old change region**, $\mathcal{N}_2$ as the **new change region**, $\mathcal{M}$ as the **old net**, and $\mathcal{M}'$ as the **new net**. Another entity which, from a modeling stand point will be part of the change, is the (labelled) sequence w of all activities which took place prior the change. This sequence will be referred to as the **pre-change** sequence. Its role is crucial for the correctness criterion (to be introduced in the next section).

The question as to how the change regions are selected, remains unsettled. Typically the old change region contains all the activities that are affected by the change (e.g. deleted, reorganized etc...), and is defined as being the smallest net containing these activities. This means that when selecting the old change region, places (with their tokens) connected to the affected activities as well as the connecting edges are part of the old change region. The next important issue relevant to the selection process is how the old change region is connected to the its context (i.e. the portion not affected by the change). More formally, this can be rephrased as what type of commnunication or interaction exists between the old change region and its context. In our case, the old change region is connected to the context through its interface. Thus the communication is restricted to token exchange through the interface. Note here that it is always possible to select appropriatly the old change region. For, in the worst case the old change region can be the whole net. The new change region embodies the changes made to the procedure and is also a marked net. Here the marking is viewed as a **token transfert** from the old change region. As we shall see, this transfert can result in the creation of new tokens or the destruction of tokens. However, the interface-marking is preserved.

When all these conditions are satisfied, the replacement may take place, resulting in a new marked net $\mathcal{M}' = (M'; m')$. Intuitively, $\mathcal{M}'$ is obtained from $\mathcal{M}$ by removing $\mathcal{N}_1$ from $\mathcal{M}$ and "plugging" $\mathcal{N}_2$ in the remaining net by using the interface as sockets . The pair $\delta = (\mathcal{N}_1, \mathcal{N}_2)$ is called a **replacement pair applicable** to $\mathcal{M}$, and the marked net $\mathcal{M}' = (M'; m')$, denoted $\mathcal{M}[\mathcal{N}_1 \longrightarrow \mathcal{N}_2]$, is referred to as the **replacement of $\mathcal{N}_1$ by $\mathcal{N}_2$ in $\mathcal{M}$**. Formally,

Definition 6.1 *A* **dynamic change** *is a system $C = (w, \mathcal{M}, \delta, \mathcal{M}')$ where:*
- *$\mathcal{M} = (M; m)$ and $\mathcal{M}' = (M'; m')$ are flow nets.*
- *$\delta = (\mathcal{N}_1, \mathcal{N}_2)$ is a replacement pair applicable to $\mathcal{M}$ such that $\mathcal{M}' = \mathcal{M}[\mathcal{N}_1 \longrightarrow \mathcal{N}_2]$.*
- *$w \in \underline{Lang}(M, i, m)$, for some $i \in \mathrm{N}^+$.*

Example 6.1 *Returning to our example of the office procedure for order processing, the first change reflects*

a new organizational policy under which it has been decided to initiate the Credit-Check and the Inventory-Check at the same time. The old version, referred to as $Order_1$ is depicted in Figure[1], the new version, referred as $Order_2$, is depicted in Figure[3], the change regions are depicted in Figure[3], and the pre-change sequence w_1 = Order-Entry.Inventory-Check. This change will be referred to as $Change_1$.

The execution resumes in $Order_2$ and sometime later another change is carried out. Here, the organization decides to initiate the shipping activity after the billing activity.

Example 6.2 *The old net for this change, referred to as $Order_3$ is depicted in Figure[4], the new net, referred to as $Order_4$, is depicted in Figure[6], the change regions are depicted in Figure[5] and the pre-change sequence is w_2 = Order-Entry.Credit-Check.Inventory-Ckeck.Evaluation.Approval.Shipping. This change will be referred to as $Change_2$.*

The dynamic change we have described earlier can be termed as **immediate**. In other words, whatever change an organization decides to do takes effect immediatly. As opposed to **delayed** change which we propose to describe next. The delayed change is called synthetic cut-over change. Here, both the old and the new change regions are maintained in the new procedure. This ensures that tokens already in the old change region will continue their progression as if the change did not take place immediatly (which justifies the attribute delayed). However tokens evolving in the context of the old change region will never enter the old change region (but possibly new change region); that is to say that in view of these tokens the change is immediate.
The motivation behind this class of changes will become clear later. We will show that in some cases, delayed change is much more safer that immediate change. In other words, it is possible to guarantee correctness for delayed change whereas this is not the case for the immediate change. Formally,

Definition 6.2 *Let $C = (w, \mathcal{M}, \delta, \mathcal{M}')$ be a change over Σ where $\delta = (\mathcal{N}_1, \mathcal{N}_2)$. The* **synthetic cut-over change** *(SCOC for short) associated with C is the change $\hat{C} = (w, \mathcal{M}, \hat{\delta}, \hat{\mathcal{M}'})$, denoted $\underline{scoc}(C)$, such that $\hat{\delta} = (\mathcal{N}_1, \underline{fuse}(\mathcal{N}_1, \mathcal{N}_2))$.*

Example 6.3 *Figures[7-9] depict the components of the SCOC associated with $Change_2$. Note here that any new job which enters the new net (depicted in Figure[9]) if it is not rejected, will go through billing and shipping. Whereas the change did not really take place for the token inside the change region.*

7 Dynamic Change Correctness

In dealing with the problem of correctness of the change in workflow systems, we learned above all that there is no single good notion of correctness and more importantly, different organizations are likely to be concerned with different notions of correctness. Three key issues have been crucial in defining our notion of correctness. They are:

• **fault prevention**: Changing a non-faulty system into a faulty one should be considered as incorrect. A system is faulty if it cannot reach a terminal marking. In general, managers are reluctant to replace productive systems by non-productive ones.
• **cancel all**: Any change in which both the old net and the new net are in an initial marking should always be correct provided that the fault prevention property is satisfied. This type of change is referred to as **system replacement**. The rationale behind this argument is that system replacement corresponds to the case where an organization decides to void whatever is in progress prior the change, make the change and restart the system.
• **consistency**. This issue is related to the meaning of the change itself. Here, we are in situation where some in-progress work (modeled by the pre-change sequence w) is resumed in a new environement (modeled by the new net). At this point we are faced with two possible situations. First, w' is intended to effectively continue the work initiated through w. Second, the in-progress work is effectively switched to a new environment, namely the new net, which means that, according to our model, the **hybrid** sequence ww' is a labelled execution sequence of the new net. Formally,

Definition 7.1 *Let $C = (w, \mathcal{M}, \delta, \mathcal{M}')$ be a change where $\mathcal{M} = (M; m)$, $\mathcal{M}' = (M'; m')$ and let w be an element of $\underline{Lang}(M, \underline{i}, m)$ for some $i \in \mathbf{N}^+$. C is said to be* **correct** *iff for every $j \in \mathbf{N}^+$, the following properties hold:*
• $\underline{Lang}(M, m, \overline{j}) \neq \emptyset \Rightarrow \underline{Lang}(M', m', \overline{j}) \neq \emptyset$.
• $\forall w' \in \underline{Lang}(M', m', \overline{j})$, **either**
$w' \in \underline{Lang}(M, m, \overline{j})$ **or** $ww' \in \underline{Lang}(M', \underline{i}, \overline{j})$.

Example 7.1 *Concerning the change $Change_1$ of Example 6.1, all hybrid sequences are execution sequences of the new net ($Order_2$), which means that it is correct.*

Example 7.2 *The change $Change_2$ of Example 6.2 is not correct. Note that the only continuation of $Order_4$ is w' = Archiving. It is clear that neither w' is a continuation of $Order_3$ nor the sequence ww' = Order-Entry.Credit-Check.Inventory-Check.Evaluation.Approval.Shipping.Archiving is an execution of $Order_4$ (a good has been shipped to a customer, yet the customet has not been billed for it). But,*

$\underline{scoc}(Change_2)$ *is correct. This is because all hybrid sequences are execution sequences of the old net* $Order_2$. *This example shows that in some cases delayed change is safer than immediate change.*

8 Up and Down Sizing

In this section we deal with two particular types of changes which involve decomposable transactions. The downsizing is a property of the change regions (assumed to be here decomposable transactions). It stipulates that every elementary execution of the new change region is also an elementary execution sequence of the old change region; in other words the new change region can "do less" than the old region. The upsizing property is the dual counterpart of the downsizing property: here, every elementary execution sequence of the old change region is also an elementary execution of the new change region, or equivalently the new region can "do more" than the old region.
In order to formalize these notions, we will make use of the formentioned terminology. A replacement pair involving decomposable transactions is referred to a **DT-replacement pair** and a change involving a DT-replacement pair is referred to as **DT-change.**

Definition 8.1 *Let* $\delta = (\mathcal{N}_1, \mathcal{N}_2)$ *be a DT-replacement pair where* $\mathcal{N}_1 = (N_1; m_1)$ *and* $\mathcal{N}_2 = (N_2; m_2)$. *Then* δ *has*

- *the* **downsizing** *property iff*
  $\underline{Lang}(N_2, \underline{1}, \overline{1}) \subseteq \underline{Lang}(N_1, \underline{1}, \overline{1})$.
- *the* **upsizing** *property iff*
  $\underline{Lang}(N_1, \underline{1}, \overline{1}) \subseteq \underline{Lang}(N_2, \underline{1}, \overline{1})$.

Example 8.1 *Note here that* $Change_1$ *has the upsizing property and that* $Change_2$ *has the downsizing property.*

Our main results, the proof of which can be found in [8], state that a dynamic change C with the upsizing property can always be carried out correctly, wheareas if C has the downsizing property, then its delayed version $\underline{scoc}(C)$ is always corret. Formally,

Theorem 8.1 *Let* $C = (w, \mathcal{M}, \delta, \mathcal{M}')$ *be a DT-change such that* δ *has the downsizing property. Then* $\underline{scoc}(C)$ *is correct.*

Theorem 8.2 *Let* $C = (w, \mathcal{M}, \delta, \mathcal{M}')$ *be a DT-change where* $\delta = (\mathcal{N}_1, \mathcal{N}_2)$, $\mathcal{N}_1 = (N_1; m_1)$ *and* $\mathcal{N}_2 = (N_2; m_2)$. *If* δ *has the up sizing property then there exists a marking* $\bar{m}_2$ *of* N_2 *such that the change* $\bar{C} = (w, \mathcal{M}, \bar{\delta}, \bar{\mathcal{M}}')$ *with* $\bar{\delta} = (\mathcal{N}_1, (N_2, \bar{m}_2))$, *is correct.*

The application of these results has already been demonstrated in the previous examples: we saw that $Change_2$ had the downsizing property and that $\underline{scoc}(Change_2)$ was correct.

9 Conclusion

In this paper we have introduced a mathematical formalism to model and analyze dynamic structural change within workflow procedures. As an example, we have defined a class of change called synthetic-cut-over which maintains correctness when downsizing occurs. This work is far from being complete, and many questions remain unanswered. Examples of such questions include (and are not limited to) the investigation of different notions of correctness, and the existence of a complete set of elementary changes that can, under some conditions, guarantee correctness and that are powerful enough to model complex changes.

Aknowledgements We wish to thank the anonymous referees for their helpful comments

References

[1] Bender, E. *Workgroup Computing,* PC World Magazine, January 1995 issue, pp.225-244.

[2] Bair, J. (Co-editor), *Office Automation Systems: Why Some Work and Others Fail,* Stanford University Conference Proceedings, Stanford University , Center for Information Technology, 1981.

[3] Bair, J. *Contrasting Workflow Models,* Proceedings of GroupWare'93, pp. 229-237.

[4] Bracchi, G. and Pernici, B. *The Design Requirements of Office Systems,* ACM Transactions on Office Information Systems, 2, 2, April, 1984, pp. 151-170.

[5] Bull Corporation, *FlowPath Functional Specification,* Bull S. A., Paris, France, September, 1992.

[6] Chang, S.K. and Chan W.L. *Transformation and Verification of Office Procedures,* IEEE Transactions on Office Information Systems, Vol. 6, No 2, 1988.

[7] Croft, W. B. and Lefkowitz, L. S. *Task Support in an Office System,* ACM Trans. Office Information Sys 2, 3, July, 1984, pp. 197-212.

[8] Ellis, C., Keddara, K., Rozenberg, G., *The Modeling of Dynamic Change Within Workflow Systems,* to apper as a technical report.

[9] Fischer, L. and White, T. (eds) *New Tools for New Times: The Workflow Paradigm* by Fischer, L. and White, T. (eds) Future Strategies Inc, Alameda, CA. 1994.

[10] Hirschheim, R. A. *Office Automation: A Social and Organizational Perspective,* John Wiley and Sons, 1985.

[11] Keddara, K., Ellis, C., Rozenberg, G., *The Modeling of Dynamic Change Within Workflow Systems*, to apper as a technical report.

[12] Osterweil, L., *Automated Support for the Enactment of Rigorously Described Software Processes,* Proceeding of the Third International Process Programming Workshop, 1988, pp.122-125. IEEE Computer Society Press.

[13] Reisig W., *Petri Nets: An Introduction.* EATCS Monographs on Theoretical Computer Science, Springer Verlag, Heidelberg (1985).

[14] Robinson, M. *Design for Unanticipated Use ...,* Proceedings of the Third European Conference on CSCW-ECSCW'93, edited by Simone, C. et al., Kluwer Academic Publisher, Sept. 1993.

[15] Suchman, L. A. *Office Procedure as Practical Action: Models of Work and System Design,* ACM Transactions on Office Information Systems, 1, 4, October, 1983, pp. 320-328.

[16] Vernadat,F., Leva, A.D., Giolito, P. *Organization and Information System Design of Manufacturing Environments: the new M* Approach,* Computer-Integrated Manufacturing Systems, Vol. 1, No 2, May 1988.

[17] Wainer, J. and Ellis, C. A. *Goal Based Models of Collaboration,* Collaborative Computing Journal, Vol. 1, No. 1, June 1994.

Figure 1

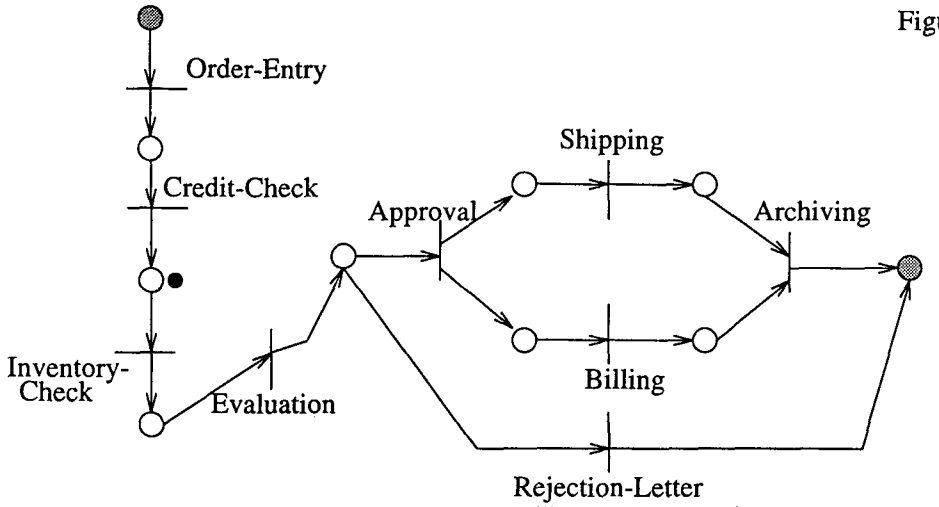


Figure 2

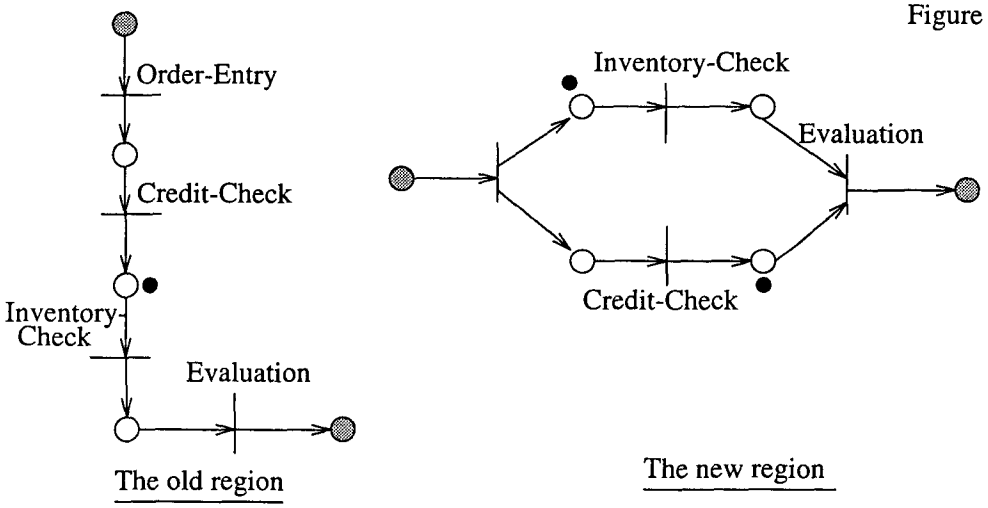


Figure 3

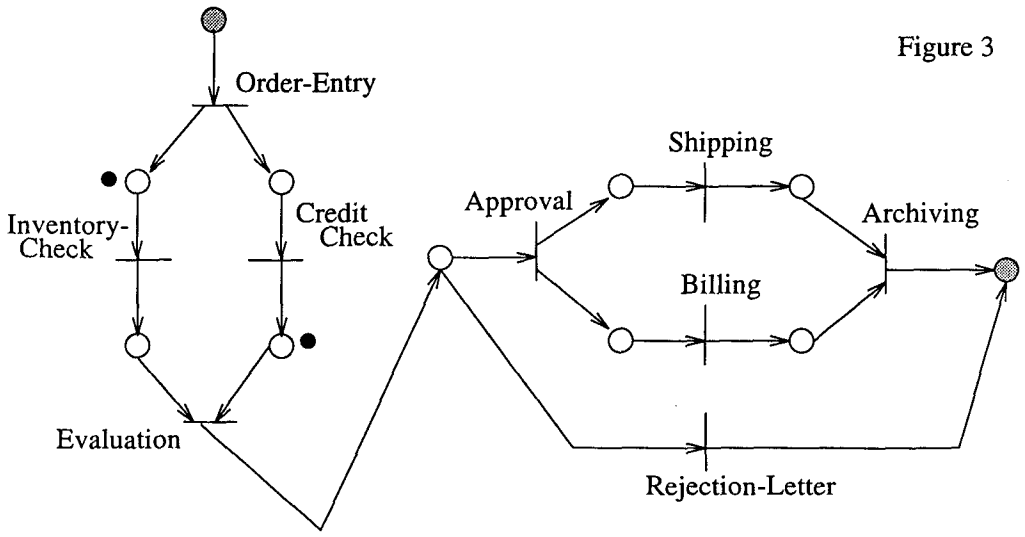


Figure 4

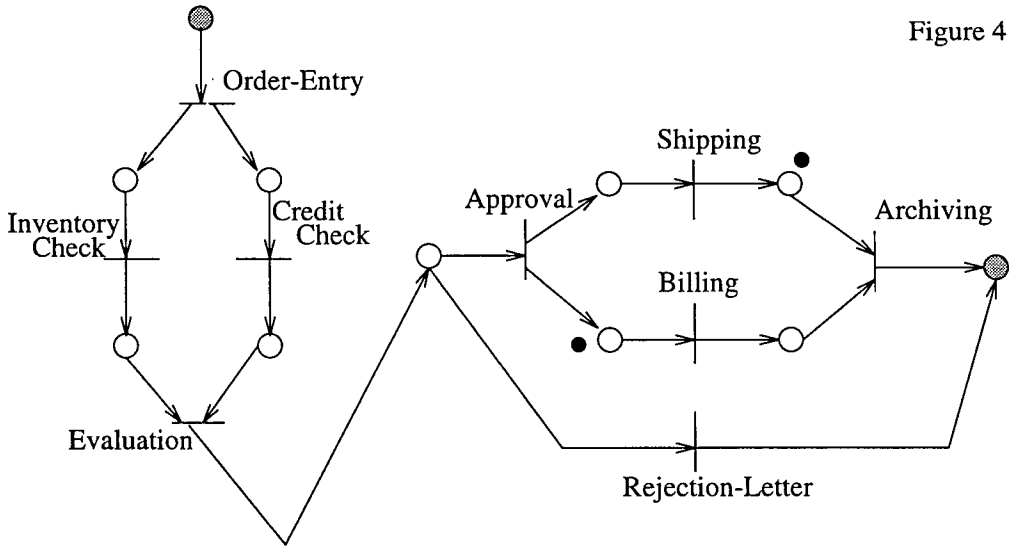


Figure 5

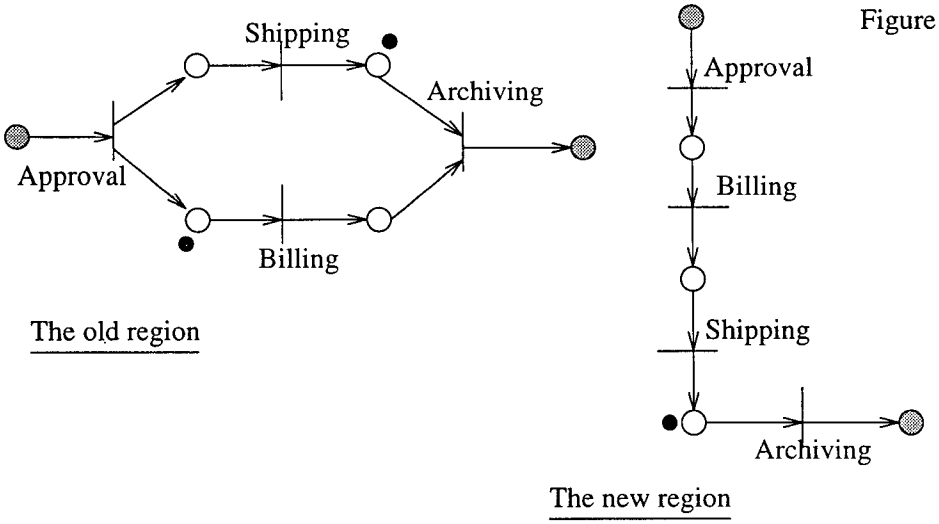
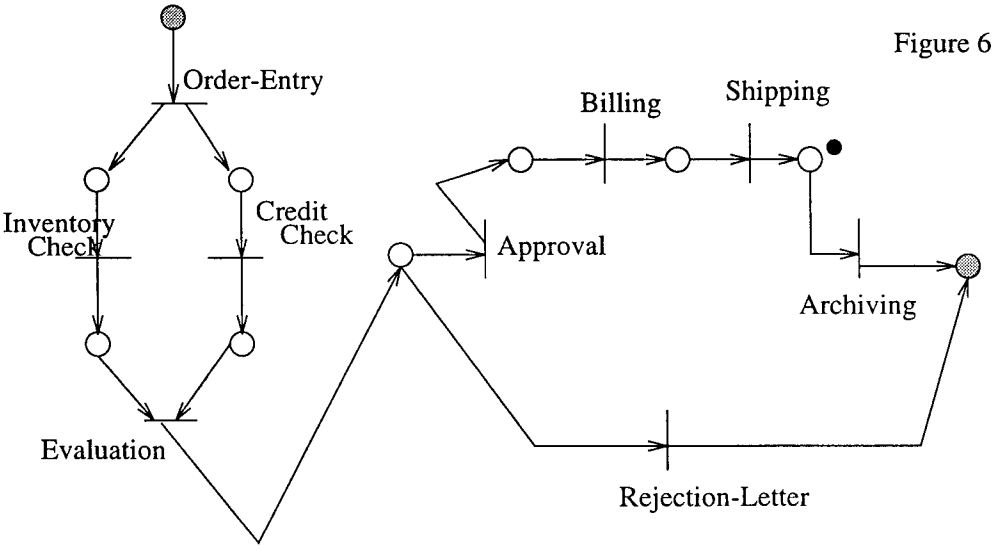


Figure 6



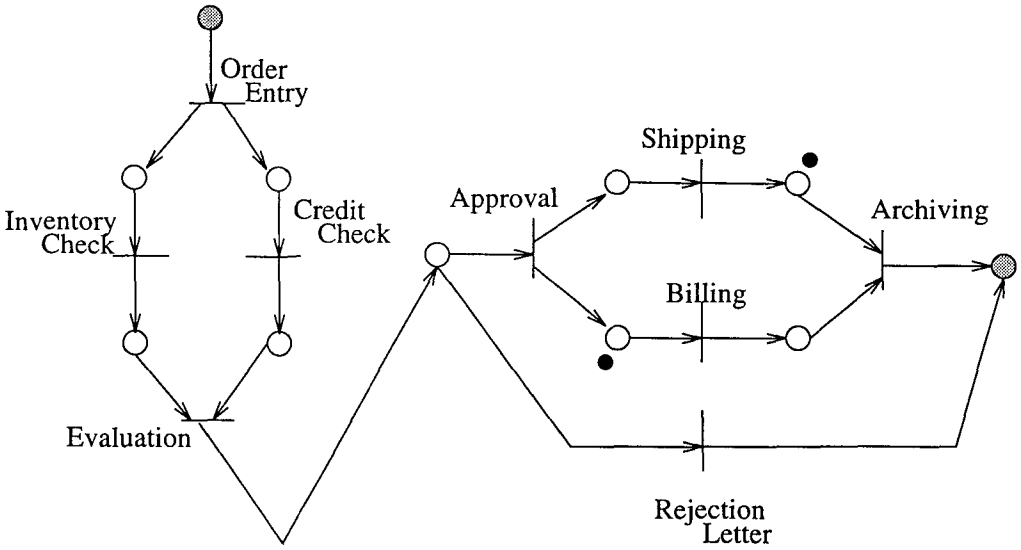


Figure 7

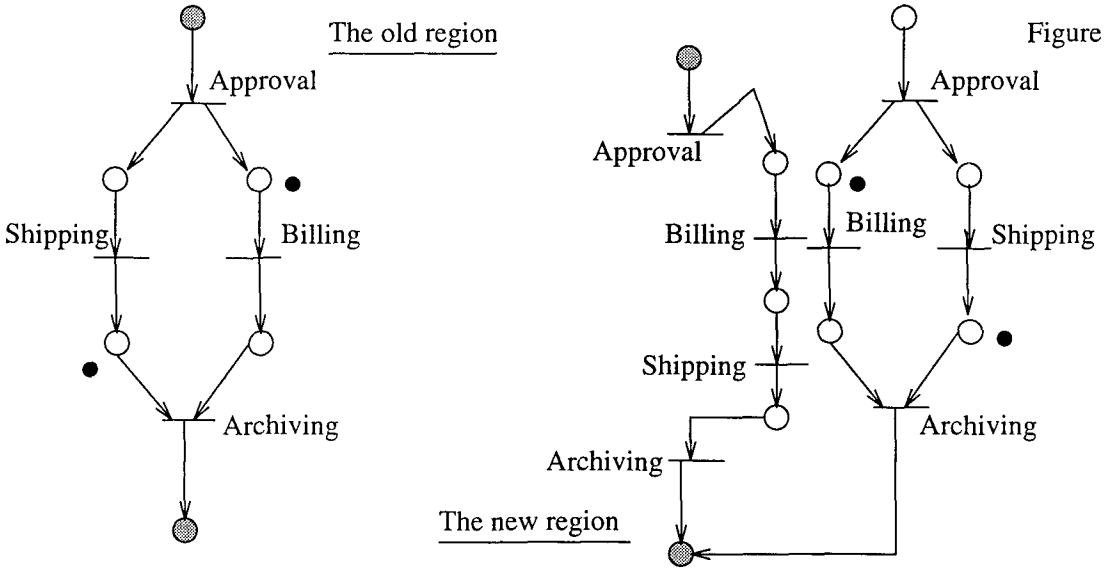


Figure 8

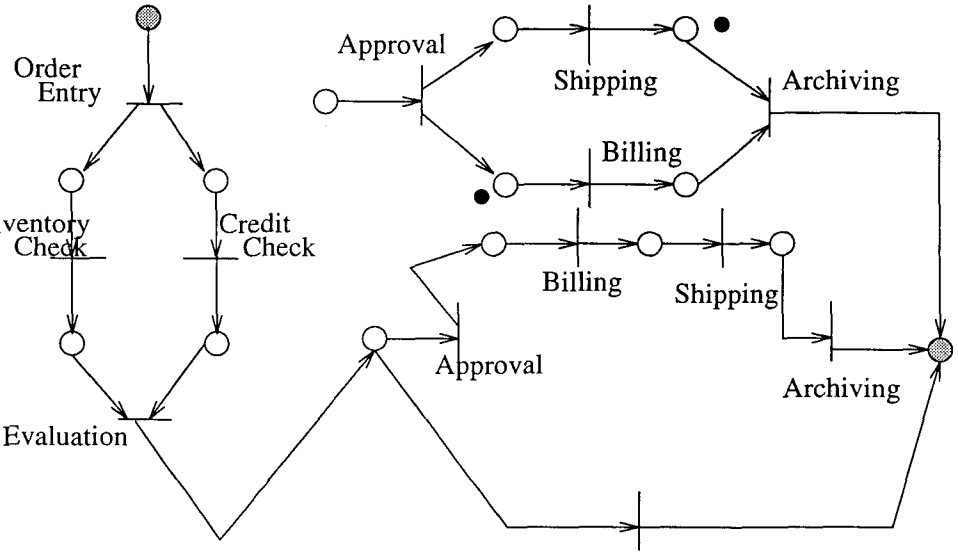


Figure 9