Dynamic Change within Workflow Systems

Clarence A. Ellis and Karim Keddara

CU-CS-667-93 August 1993

University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

Dynamic Change within Workflow Systems

Clarence A. Ellis
Karim Keddara
Department of Computer Science
University of Colorado
Boulder, CO 80309-0430
e-mail: {skip,karim}@cs.colorado.edu

Keywords: Wokflow Systems, Information Control Nets, Dynamic Change.

1 Abstract:

This paper is concerned with dynamic change in workflow systems. Workflow systems, many of which have been recently introduced onto the marketplace, are computer based groupware systems which help organizations to specify, analyze, manage, and execute their procedural work steps. Within this domain, one of the problems which is severely in need of study and solution is the dynamic structural change problem that frequently occurs in practice. Since organizations are continually evolving, there is frequently a need for structural change of procedures, such as adding or deleting a step of the procedure, or changing the order in which steps of a procedure are executed. One option is to flush the system, so that no work is in progress, and then make the structural changes. This option is inconvenient, resource wasteful, and in some environments, infeasible. Another option that we explore in this paper is to perform the structural changes dynamically, while the system is in execution. How can the system assist in verifying that these (sometimes very complex) dynamic changes maintain correctness?. This paper, in the context of the ICN (Information Control Net) model of workflow, defines and addresses this dynamic change problem. We note that dynamic change is a large and pervasive issue which surfaces within groupware, as well as within software engineering, manufacturing, and numerous other domains; thus, its solution may be of interest to many disciplines. Several categories of change are identified within which ICN analysis can be helpful. As a normal form for change, we define "synthetic cut-over change," and show that use of our synthesized change graph guarantees correctness. We conclude the paper by summarizing results obtained, noting limitations encountered, and mentioning some future research directions.

2 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. These latter tools are called groupware. *Groupware* is defined as "systems that support groups engaged in a common task or goal, and that provide an interface to a shared environment." [Ellis91] The potential benefits and pitfalls of groupware have been discussed in conferences, tutorials, and journals concerned with the new area of computer supported cooperative work (CSCW) [Grudin88, Poltrock91]. As opposed to much of the previous generation of office information systems, this literature addresses the inherently interdisciplinary nature of groupware. To successfully implement groupware in an organization requires well grounded technological development, with careful attention paid to the social and organizational environment into which the technology is being embedded.

Many groupware products have recently been introduced to the market [Dyson92]. A few of these products capture knowledge of the organizational activity that they are assisting, but the vast majority do not. 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." [Bull92] The system contains two basic components: the first component is the workflow model, which enables administrators and analysts to define procedures and activities, analyze and simulate them, and assign them to people. Most workflow products have no model, so this component is called the "specification module"; usage of this module is typically completed before the flow of work tasks actually begins. Our research explores the hypothesis that a model of coordination is a useful entity in all phases of the workflow cycle. In section 4, we present the Information Control Net (ICN) as our workflow model. It allows us to precisely and mathematically define notions of correctness and dynamic change.

The second component is the workflow execution module (the workflow 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. Some of these steps may be executed in parallel; some executed automatically by the computer system. The execution interface is utilized for all manual steps, and typically presents forms on the electronic desktop of appropriate workers (users.) The user fills in forms with the assistance of the computer system. Various databases and servers may be accessed in a programmed or ad hoc fashion during the processing of a work step. In section 2, this second component of workflow is further explained.

How do the first and second components relate? Our research explores the hypothesis that the specification and execution modules need to be tightly interwoven. This hypothesis 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

In order to make structural changes as above within a workflow system context, it is typically and unfortunately necessary to suspend or abort the work in progress within the execution module, and start up the specification module to make the changes to the specification. Then after the change, the specification module is terminated, and once again, the execution module is started. This is an inefficient, and ineffective procedure because many organizations find it very unproductive, and sometimes impossible, to shut down all activity 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 process. Today, organizations usually do not solve this problem, they cope, evade, or "muddle through." Using the ICN definition and example presented in section 4, we show, in section 5, a simple example of dynamic structural change, and a typical correctness problem that can occur when dynamic structural change is attempted. Section 5 also presents the mathematical arguments and theorem of correctness.

By combining the first and second components of workflow, the model is constantly available and change can potentially occur dynamically if the correctness and consistency problems of dynamic change can be solved. Thus, even with these components combined, we do not know how to smoothly and correctly handle the myriad of changes which are constantly happening. Although there is considerable literature addressing workflow, office modelling, and business re-engineering, the problem of dynamic structural change has not been generally addressed and solved. In section 3, we discuss related work in the literature and in research labs.The conclusion is that in large organizations around the world, dynamic change is an ad-hoc and risky event.

The history of workflow application in corporate America has been mixed; more systems have silently died than been successful [Bair81]. It is found that organizations succeed only if people creatively violate, augment, or circumvent the standard office procedures when appropriate. People tend to work through goals rather than through procedures [Li90]. Opportunity exists for a leap forward in productivity, effectiveness, and satisfaction when workflow systems successfully incorporate and utilize knowledge of goals, constraints, and the social and organizational context into which they are embedded. The authors are associated with an ongoing research project, the Collaboration Technology Research Group, at the University of Colorado, which is actively researching systems and models to enhance workflow in these directions. Today, these types of workflow systems are only a vision of the future. Structured procedural work frequently has unstructured components. The mechanisms to help people do their necessary problem solving and exception handling are not available in today's workflow systems. The research reported in this paper is concerned with dynamic change; an issue which must be addressed for this vision to become a reality.

2.1 Workflow Concepts and Architecture:

2.1.1 Definition (Workflow):

A *workflow* system is an application level program which helps to define, execute, coordinate and monitor the flow of work within organizations or workgroups.

In order to do this, a workflow system must contain a computerized representation of the structure of the work procedures and activities.

Many types of office work can be described as structured recurring tasks (called *procedures*) whose basic work items (called *activities*) must be performed by various people (called *actors*) in a certain sequence. The power of workflow systems lies in their computerized representation of these procedures, and activities. This section of the paper describes the basic terminology and capability of workflow; much more power and utility is possible once this procedural representation is available within the computer system.

A particular workflow application is created by specifying to the workflow system a set of procedures and activities which are performed within an organization or workgroup. This is the first step toward computerized workflow; the goal is to enhance the efficiency and effectiveness of the office work.

2.1.2 Definition (Procedure):

A *procedure* is a predefined set of work steps, and partial ordering of these steps. A work step consists of a header (identification, precedence, etc.) and body (the actual work to be done.)

Examples include the "order processing procedure" within an engineering company, and the "claims processing procedure" within an insurance company. Both of these are relatively standardized and structured, and each can be described by a sequence of steps. Workflow also attempts to assist in less structured work tasks. Different steps of a procedure may be executed by different people or different groups of people. In some cases several steps of a procedure may be executed at the same time or in any order. In general, we therefore define a procedure to be a partially ordered set of steps rather than a totally ordered set. We also define workflow procedures in such a way that loops are allowed. Procedures typically have attributes, such as name and responsible person, associated with them.

2.1.3 Definition (Activity):

An *activity* is the body of a work step of a procedure. An activity is either a compound activity, containing another procedure, or an elementary activity.

An *elementary activity* is a basic unit of work which must be a sequential set of primitive actions executed by a single actor. Alternatively, an elementary activity may be a non-procedural entity (goal node) whose internal we do not model within our structure. An activity is a reusable unit of work, so one activity may be the body of several work steps. For example, if "order entry" and "credit check" are procedures, then the activity "send out letter" may be an activity in both of these procedures. In this case, these are two distinct steps, but only one activity. An activity instance associated with the body of a particular work step is called a work step activity.

Activities typically have attributes such as description and mode associated with them. An activity has one of three modes. Some work step activities may be automatically executed (automatic mode,) some completely manual (manual mode), and some may require the interaction of people and computers (mixed mode). For example, if the procedure is "order equipment" then there may be work steps of:

1. order entry
2. credit check
3. billing
4. shipping

This level of detail of description is typically adequate for an engineering manager, but is not enough detail for an order administrator. The order administrator would like to look inside of the work step called order entry, and see a procedure that requires logging data and filling out of a form. Thus, the body of this step is itself a procedure with work steps of:

1.1. log name and arrival time

1.2. fill out the order form

1.3. send out acknowledgment letter

Furthermore, step 1.2 of filling out the order form may itself consist of work steps to fill out the various sections of the form. This example shows that it can be useful to nest procedures within procedures. Thus, a work step body has been defined to possibly contain a procedure. Work steps typically have attributes, such as unique identifier and executor, associated with them.

By definition, a workflow system contains a computerized representation of the structure of procedures and activities. This also implies that there is a means for someone (perhaps a system administrator) to specify and input descriptions of procedures, activities, and redwings into the computer. These specifications are called scripts. In the next section of this paper, we introduce ICNs as a scripting language.

2.1.4 Definition (Script):

A *script* is a specification of a procedure, an activity, or an automatic part of a manual activity. The composition or building of this script from available building blocks is called scripting.

Once procedures and activities have been defined, the workflow system can assist in the execution of these procedures. We separate the concept of the static specification of a procedure (the template) from its execution.

2.1.5 Definition (Job):

A *job* is the locus of control for a particular execution of a procedure. In some contexts, the job is called a work case; if a procedure is considered a Petri net, then a job is a token flowing through the net. If the procedure is an object Class, then a job is an instance.

In our example, if two customers submit two orders for equipment, then these would represent two different jobs. Each job is a different execution of the procedure. If both jobs are currently being processed by the order entry department, then the state of each job is the order entry state. Jobs typically have parameters such as state, initiator, and history associated with them.

Because of the ever changing and sometimes ad hoc nature of the workplace, it is important for workflow systems to be flexible, and have capabilities to handle exceptions. Many procedures which appear routine and structured are, in reality, highly variable, requiring problem solving and creative exception handling. Exception handling is one form of dynamic change; we rigorously define dynamic within a later section of this paper.

Another workflow concept that helps address these issues is the indirect association of people with activities via the concept of roles.

2.1.6 Definition (Role):

A *role* is a named designator for an actor, or a grouping of actors which conveniently acts as the basis for access control and execution control. The execution of activities is associated with roles rather than end users.

Thus, instead of naming a person as the executor of a step, we can specify that it is to be executed by one or more roles. For example, instead of specifying that Michael executes the order entry activity, we can specify that

1. the order entry activity is executed by the order administrator, and
2. Michael is the order administrator.

There may be a very large number of work steps in which Michael is involved. When Michael goes on vacation, it is not necessary to find and change all procedures and work steps involving Michael. We simply substitute Michael's replacement in the role of order administrator by changing step 2. to

2. Robert is the order administrator.

A role may be associated with a group of actors rather than a single actor. Also, one actor may play many roles within an organization. If there are many order administrators within our example, then these can be defined as a group, and it is easy to send information to all order administrators. In this case, an option may be available to "send to all" or alternatively, "send to any" administrator, and the system might use some scheduling algorithm to select one. Other flexible scheduling algorithms are possible, including the notification of all members of the group that a job is available, and allowing the first responder to handle the job. In this document, we use the term actor to refer to a person, a group, or an automated agent. For example, the credit check activity in our example is really executed by the credit department, not by any single person. And the printing operation is really executed by one of many print servers that might be actors with the role of "printer".

2.1.7 Definition (Actor):

An *actor* is a person, program, or entity that can fulfill roles to execute, to be responsible for, or to be associated in some way with activities and procedures.

Access attributes or capabilities may be associated with actors and with roles. Other attributes, parameters and structures can be associated as needed. For example, the role of manager is perhaps only played by Michael within the order entry department. thus a parameter of the role may be the group within which this role applies.

In summary, we have briefly presented a definition of workflow together with explanations of the concepts of procedure, step, activity, job, script, role, and actor.

2.2 Conceptual Architecture:

In the remainder of this section, we present the conceptual architecture of a generic workflow system using the entity-relationship (E-R) model. the architecture builds upon the general concepts introduced in the previous subsection. It lays out the workflow system conceptual entities and their relationships.

The entity-relationship model is a high level semantic model using nodes and arcs; this model has proven useful as an understandable specification model, has been implemented within E-R databases, directly parallels some object oriented concepts, and has a well known direct mapping into a relational database.

In the E-R model, objects of similar structure are collected into entity sets. The associations among entity sets are represented by named E-R relationships which are either one-to-one, many-to-one, or many-to-many mapping between the sets. The data structures, employing the E-R model, are usually shown pictorially using the E-R diagram. An E-R diagram depicting the conceptual architecture of a workflow system is shown in Figure 1. A labeled rectangle denotes an entity set; a labeled arc connecting rectangles denotes a relationship between the corresponding entity sets.

In Figure 1, the box labeled procedure denotes an entity set of procedures that may actually be a table of procedure names and their attributes. Likewise, activity may be a table of activity names and their attributes. There is an arc connecting these two boxes because there is a relationship called "part-of" between these two entity sets. Some elements in the activity set are steps of (or parts of) some procedures. This arc is labeled with the relationship name, and a denotation of M and N indicating that this is a many to many relationship. Therefore, a procedure can contain many activities, and an activity can be part of more than one procedure. the arc joining the activity box to itself labeled precedence tells which activities may precede which others.

Since the diagram specifies that this is a many to many relationship the procedure scripting facility supports the specification of conjunctive and disjunctive precedence relations. For any activity labeled conjunctive, any specification of immediate successors denotes activities which all directly follow the completion of the given activity; specification of immediate predecessors denotes activities which must all complete before the given activity can begin. some activities will be labeled disjunctive. OR-out from some activity means that out of the many immediate successor activities, we select only one to actually execute. Similarly, OR-in means that only one of the activities which immediately precede the given activity must complete before it can begin. thus, any partial ordering of activities using sequencing, and these AND/OR constructs can be specified and supported using workflow.

Other entities shown in Figure1 are job and data.A job, which can be considered to be flowing through a procedure, has a state at any instant which is denoted by one or more current activities being executed by the job. The relationship "state-of" captures this state. This relationship gets updated by the system each time that a job moves from one activity to another. This is a many to many relationship, so one job may be executing within several activities in parallel, and one activity may be simultaneously serving several jobs. Similar considerations hold for the data entity which refers to the application data which is accessed by the various activities. People are connected into the system directly if they are listed in the "actor" entity set. Thus, people are players of roles, and roles are designated as the executors of activities.

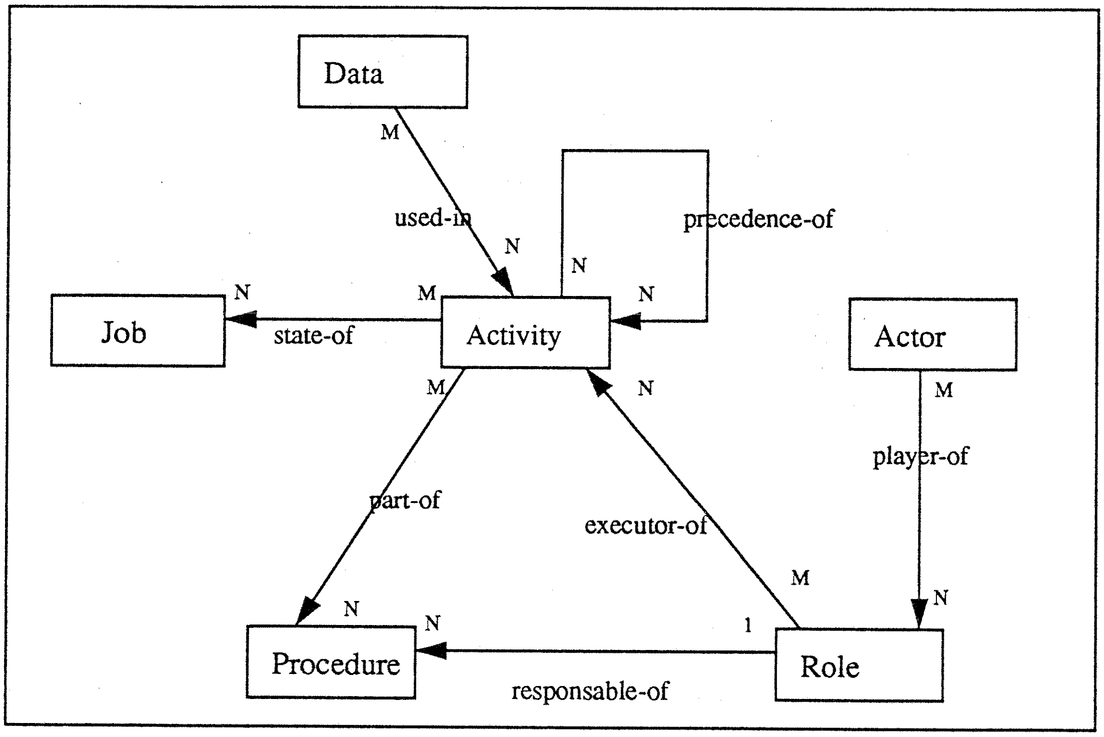


Figure 1: Workflow Conceptual Architecture

3 Related Work:

There has been considerable work which addresses workflow systems. Some of the beginnings in this area come from one of the author's early work on Officetalk / ICNs in the 1970s [Ellis80]. Also, GMD has implemented several versions of Domino [Kriefelts84], a Petri net based prototype office information system. Usage reports detail numerous problems and reasons for user rejection of the system -- this typifies problems of current workflow. Other workflow efforts include the Xerox "Collaborative Process Model" [Sarin91], Polymer at the University of Massachusetts [Croft84], Prominand [Karbe91], Role Interaction Nets

[Rein92], and the WooRKS workflow prototype within the ITHACA ESPRIT project [Ader92]. None of these systems address the dynamic structural change problem.

Models of workflow have spanned the gamut from very informal to very formal. Informal modelling has been reported by Wynn and Suchman [Suchman83]. Early work to formalize this was presented in the thesis of Michael Zisman [Zisman77] where he developed APNs (Augmented Petri Nets) that attached production rules to specify semantics within Petri Nets. These concepts were implemented in the SCOOP system. The model UBIK represents an organization by "configurators" which perform actions by sending messages to each other [DeJong87]. The OFS model represents the flow of forms within an office; within this model, all messages, documents, letters, etc., are defined to be forms [Tsich82]. Another alternative is to model the office as a database with transactions. TEMPORA is an integrated architecture for doing business design and analysis within a database environment [Loucopoulos92].This is a small sampling of the large number of models which have been used for the modelling of offices and workflow.

Considerable effort has been put into workflow studies. Many of these have transpired in the Information Systems field and the Organizational Design field within business schools. Examples include Bair's TUMS [Bair91], Woo's SACT [Woo90], Hirshheim's model [Hirshheim85], the Society model [Ho86], Hammer's BDL and OAM [Sirbu84], and the OSSAD model [Dumas91]. Several office models have emerged from concepts of discrete mathematics. These include Petri net based workflow models [Zisman77, Holt88, Li91], and graph theory based models [Luqi90]. There is also a set of models which have emerged out of the software engineering community. These could be classified as extended flowchart/state machine notations [Harel90], project management models [Kellner91], and process programming models [Osterweil88]. Office models are reviewed and contrasted in several articles including [Ellis80] and[Bracchi84].

4 Modelling with ICNs:

Our research group at Colorado is (and has been for many years) actively researching the Information Control Net model (abbreviated ICN) for information systems analysis, simulation, and implementation. The ICN is a simple, but mathematically rigorous formalism created and designed in the 1970s specifically to model office procedures [Ellis79]. ICNs are actually a family of models which have evolved to incorporate control flow, data flow, goals, actors, roles, information repositories, and other resources [Ellis83]. ICNs have been studied in universities [Dumas91] and applied in industry [Bull92]. They have been shown to be valuable for procedures, for analysis, and for implementation. Some of the documented analyses of ICNs include throughput, maximal parallelism, reorganization, and streamlining [Cook80].

4.1 Mathematical Definitions:

The ICN family of models are structured around the fundamental observation that organizations encompass goals, resources, and constraints. Some organizations are very highly structured, with precisely defined procedures and rules; others are very loosely constructed with predominantly unstructured activities. Due to the variety of organizations, and due to the variety of questions that models may be employed to investigate, we have seen that no one model adequately addresses all aspects. Thus, we derive a family of models by selecting different types of resources and different levels of structure to incorporate in any particular member of the family. For example, an organizational model which focuses upon informal interpersonal communication must incorporate the very important resource of people, and the roles that they play in the organization. For the thrust of this paper, which investigates dynamic structural change of procedures, we use the basic "control ICN" which models partial orderings of activities and their control structures.

4.1.1 Definition (CICN):

Let A be a finite set of activity names. A *Control ICN* (CICN for short) over A is a node-labelled graph specified as a tuple G = < N, lab, start, exit, E > where:

- N is the finite set of nodes.
- lab: $N \rightarrow A \cup \{0, 1\}$ is the labeling function (assuming that neither 1 nor 0 is in A). Each node with a label in A is called an activity node, and a control node otherwise. Each control node with a label 1 is called an and-node, and or-node otherwise. Notice that two different activity nodes can carry the same label.
- start is a special activity node called the entry node.
- exit is a special activity node called the exit node.
- $E \subseteq N \times N$ is the set of edges which verifies the following:

  1- For each activity node x distinct from exit there is one and only one node y such that $(x, y) \in E$ (i.e. one outgoing edge).

  2- For each activity node y distinct from start there is one and only one node x such that $(x, y) \in E$ (i.e. one incoming edge).

  3- For every component x of G (node or edge), there exists a path in G, from start to exit, which contains x; that is to say that x is *reachable* from start and that exit is reachable from x.

Sometimes we will use G as a subscript for N, lab, start, exit and E. Thus N_G will denote N, etc...

4.1.2 Definition (Marked CICN):

Let A be a finite set of activity names. A *Marked CICN* (M-CICN for short) over A is a system < G, M > where:

- G is a CICN over A.
- M: $N \cup E \rightarrow N$ is a marking for the graph G which associates an integer specifying the number of tokens with each node and each edge of G. If $x \in N \cup E$ and $M(x) \neq 0$ then x is said to be marked with M(x) tokens. A token is a marker that may cause a node to fire.

In particular, with every CICN G we can associate an initial marking, noted M_G^0, in which only the start node carrie one token. This marking will be used by default unless otherwise is explicitly stated.

4.1.3 Example (M-CICN):

Frequently ICNs are manipulated in their graphical form. Figure 2-a shows the graphical form for an Information Control Net (ICN) depicting a procedure for order processing within a corporation. When a customer request for goods arrives, the first step is the order entry activity in which an order administrator fills out an order form. This is graphically depicted by the first (top) ellipse in figure 2-a. The ellipses thus denote activities. Arcs denote precedence, so for example, the shipping activity must complete before the billing activity can begin. After order entry is completed, inventory check and compile references activities can proceed concurrently, indicated by the gray circle labelled “and”. A corresponding second gray circle denotes the “and join” of activities. After the order evaluation activity, either shipping or rejection processing occurs.

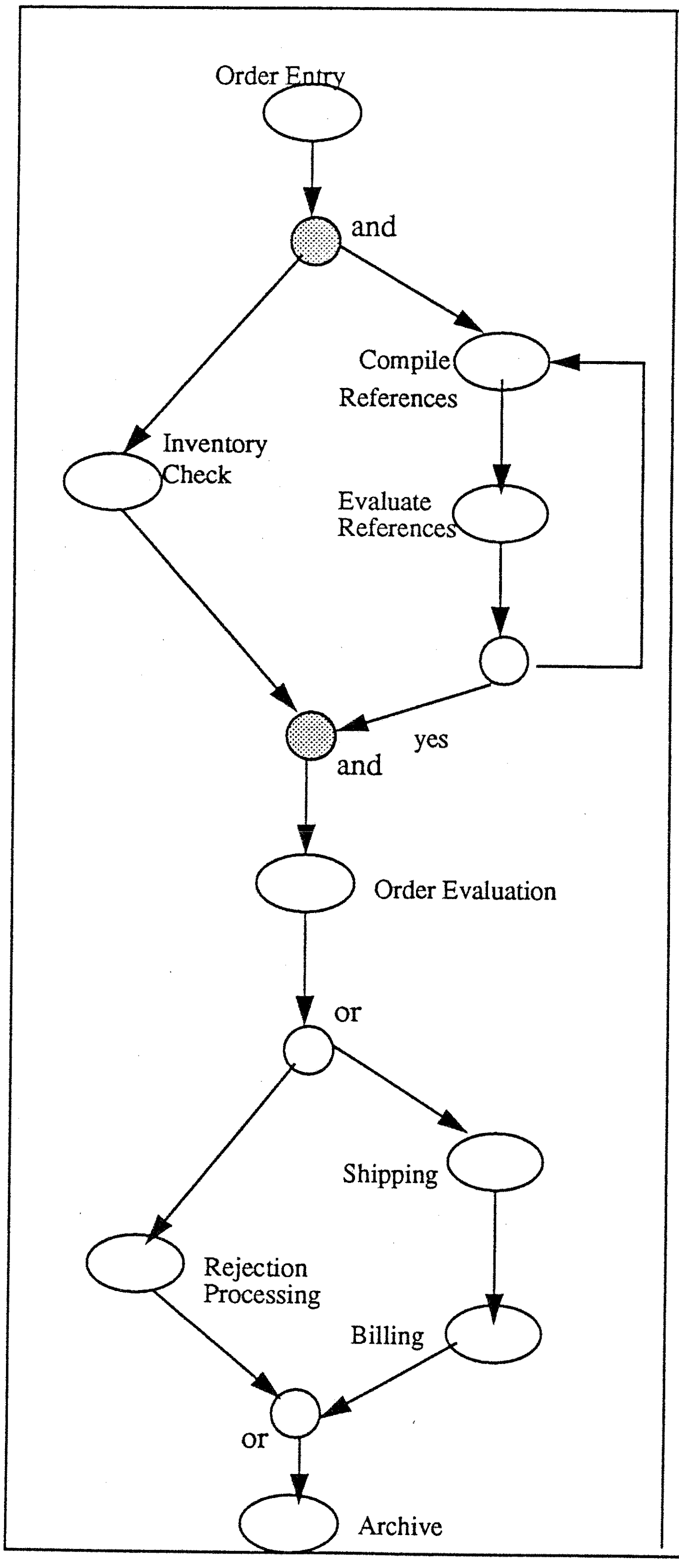


Figure 2-a: an ICN for Order Processing

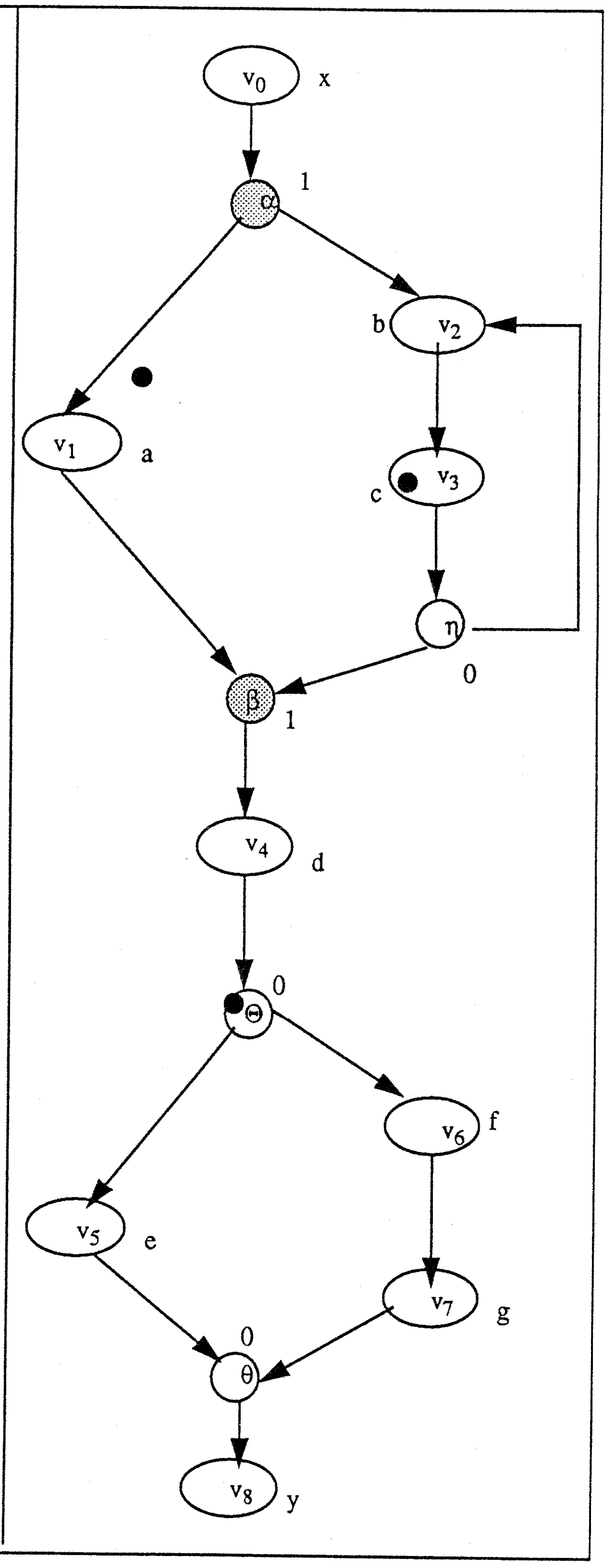


Figure 2-b: Its Formal Definition

Thus, the hollow circle "or" denotes choice or decision making. There is a corresponding "or join" hollow circle, so that the archive activity occurs after either the rejection or the billing activity is completed.

Figure 2-b depicts the formal representation of the CICN described earlier. In the graphical representation the name inside a node represents its (unique) identity, whereas the label beside a node corresponds to its label. Thus, the CICN of figure 2-b is given by:

- $A = \{a, b, c, d, e, f, g, x, y\}$.
- $N = \{v_0, \ldots, v_8, \alpha, \beta, \eta, \theta, \Theta\}$
- start = v_0, exit = v_8.
- $lab = \{(v_0, x), (\alpha, 1), (\beta, 1), (\eta, 0), (\theta, 0), (\Theta, 0), (v_1, a), (v_2, b), (v_3, c),$ $(v_4, d), (v_5, e), (v_6, f), (v_7, g), (v_8, y)\}$.

The black dots represent markings of the components (node or edge) of the CICN; in the case of figure 2-b the marking M is given as: $M = \{((\alpha, v_1), 1), (v_3, 1), (\Theta, 1)\}$.

We are now in a position to describe formally the execution of a CICN.

4.1.4 Definition (Execution of M-CICN):

Let < G, M > be a M-CICN, and let x be a node of G. Then:

- x is *S-enabled* in G under the marking M if:
  - a- x is an activity node and the incoming edge of x is marked, or
  - b- x is an and-node and all incoming edges of x are marked, or
  - c- x is an or-node and one of the incoming edges of x is marked.
- x is *T-enabled* in G under M if x is marked.
- if x is S-enabled in G under M then x can Start firing. The *S-firing* of x will add a token to x and
  - a- remove one token from the incoming edge of x if x is an activity node, or
  - b- remove one token from all incoming edges of x if x is an and-node, or
  - c- remove one token from one and only one incoming edge x is an or-node. This will be denoted as $M\,|x\rangle M'$ where M' is the newly obtained marking of G.
- if x is T-enabled in G under M then x can Terminate firing. The *T-firing* of x will remove one token from x and add a token to:
  - a- the only outgoing edge of x if x is an activity node,
  - b- every outgoing edge of x if x an and-node,
  - c- to one arbitrary but only one outgoing edge of x if x is an or-node. This will be denoted as $M\,|\bar{x}\rangle M'$ and $\bar{x}$ is said to be *barred.*

4.1.5 Definition (Computation in M-CICN):

Let < G, M> be a M-CICN, let $M_1,...,M_n$ be a sequence of markings of G and let $x_1,...,x_n$be a sequence of nodes of G (some of which are barred). Then:

- $\sigma = Mx_1M_1...x_nM_n$ is an *occurrence sequence* of <G,M> iff for every i, $1 \le i \le n$, $M_i |x_i\rangle M_{i+1}$. In this case the sequence $\sigma_G = x_1...x_n$ is called a *firing sequence* of <G,M>, and if in addition $x_n = \overline{\text{exit}}$ and the exit node is the only marked component of G under M_n then σ_G is called a *computation sequence.* σ is said to be *leading* from M to M_n and noted as $M |\sigma_G\rangle M_n$
- The language generated by < G, M> is the set defined as follows: $L(G, M) = \{\delta_G(\sigma_G) | \sigma_G$ is a computation sequence of <G,M>$\}$. where $\delta_G(\sigma_G)$ is obtained by stripping σ_G from the barred nodes (execpt the start node) and the control nodes, and applying lab to the resulting sequence of nodes. In particular L(G) is the language $L(G, M^0_G)$.

4.1.6 Example:

In the case of the M-CICN < G,M> of figure 2-b, the nodes v_3 and Θ are T-enabled, whereas the node v_1 is S-enabled. The firing sequence $\sigma_G = v_1\overline{v_3}\eta\overline{\Theta}v_5\overline{\eta}v_2\overline{v_1}$ will result in the marking defined as follows: $M' = \{((v_1, \beta), 1), (v_2, 1), ((\Theta, v_5), 1)\}$, and $\delta_G(\sigma_G) = aeb$. Moreover L(G) is given by the set $\{xwdey, xwdfgy|$ w contains one a, and w_a is a repetition of the word bc$\}$, here w_a is the word obtained from w by removing all occurences of a in w.

4.1.7 Definition (CICN Specification)

A *CICN Specification* (CICN-Spec for short) is a tuple S = < G,ψ > where G is a CICN and Ψ is a language over the alphabet A, called the *correctness criterion* of G, such that $L(G) \subseteq \psi$.

The language ψ is considered to be the correctness criterion to be used to check whether a change made in an ICN is "correct". The extra requirement means that the implementation meets (in a loose but consistent way) the specification. The reason we have selected a rather weak requirement (inclusion instead of equality) is to have more freedom in doing more modifications in ICNs. This requirement also allows us to modify arbitrarily an ICN if desired by considering the *Universal Spec* (i.e $\psi = A^*$).

For the purpose of the present paper, there is no restriction on the class of languages to which the correctness criterion must belong. Nor there is any assumptions on the means used to specify it; it can be a rational, algebraic, or logical expression. This issue will be dealt with in a forthcoming paper where complexity issues are closely examined.

5 Dynamic Change:

Change to the values of application data items is a normal type of activity that occurs in administrative information processing. However, the type of change that we are considering is structural change to the procedures and processes. Dynamic means that the change to the procedure occurs while the procedure is executing. This type of dynamic structural change is not considered "normal" by most organizations. Static change, in the ICN context, means that the execution of the procedure is halted, all tokens are removed, and the change is applied at quiescence. Static correctness means that certain assertions or constraints are not violated - it implies that we have a set of correctness criteria that hold for all tokens flowing through the ICN before the change, and also for all tokens that enter the ICN after the change is completed.

Dynamic change correctness is concerned with tokens which enter the net prior to the change and do not exit the ICN until some time after the change. Anomalous behavior can be exhibited by these tokens even if we know that the change maintains static correctness. A simple example of this is the change that includes swapping of the billing and shipping activities in the example ICN of figure 2-b. Tokens that are currently within the shipping node (node v_6) when the swap change occurs never encounter the billing activity (node v_7), so the company never gets paid for the goods that are shipped. Suppose that the correctness criterion is that all customer orders must pass through shipping and billing in some order. This anomaly occurs although the ICN before the change is correct, and the ICN after the change is correct.

In this section, we shall introduce all the definitions and terminology to deal with the problem of dynamic structural change, we will state and prove our main result. Roughly speaking, any ICN change can be turned into a correct one. The idea is to identify the primary change region in the initial ICN, i.e. the portion of the ICN where the structural change is taken place, note here that one or many tokens may be in progress in this region of the graph. The second step is to add this primary change region to the new ICN. Consequently, a token already in progress in the change region will progress as it is evolving solely in the old ICN, and that a token outside this change region will finish its progression in the new ICN. Since (by requirement) both ICNs maintain the correctness criterion, the new change (obtained by duplicating the change region) is correct.

5.1 Mathematical Definitions:

5.1.1 Definition (CICN rewriting):

a *CICN rewriting* (CICN-R for short) is a tuple $T = \langle G_1, G_2, L \rangle$ where $\langle G_1, L \rangle$ and $\langle G_2, L \rangle$ are CICN specifications.

We are assuming that both CICNs have the same correctness criterion, the main result (to be stated later) still holds if the correctness criterion is wakened in G_2.

5.1.2 Definition (Correctness):

Let $T = \langle G_1, G_2, L \rangle$ be a CICN Rewriting, let σ be an occurrence sequence of G_1 such that $M_{G_1} | \sigma_{G_1} \rangle M_1$. Then T is *correct* w.r.t σ iff there exists a (token) mapping ϕ: $G_2 \rightarrow G_1$ such that:

- 1: for every activity node x of G_1 marked under M_1, there exists an activity node y of G_2, with the same label as x, such that $x \in \phi(y)$.
- 2: for every execution sequence σ' of $\langle G_2, M_1(\phi) \rangle$, $\delta_{G_1}(\sigma) \bullet \delta_{G_2}(\sigma') \in L$. (see 4.1.5 for the definition of δ_G).

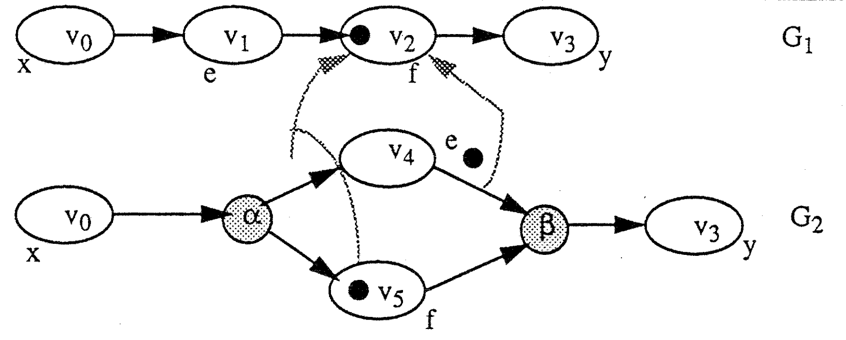


Figure 3: Parallelization of activities 1 and 2.

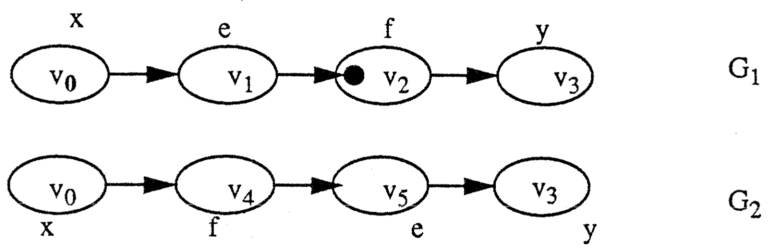


Figure 4: Swapping is not always correct.

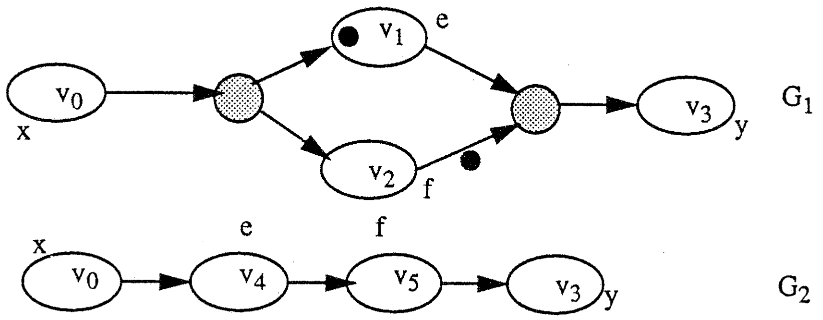


Figure 5: Sequentialization is not always correct.

In the previous definition, ϕ maps a component of G_1 (node or edge) to 0, 1, or many components of G_2, and $(M(\phi))(x) = \sum_{y=\varphi(x)} M(y)$. Informally speaking, a token in a node y of G_1 will be mapped to a token in every component x (node or edge) of G_2 such that $y \in \phi(x)$

5.1.3 Definition (Validity):

a CICN-R $T = \langle G_1, G_2, L\rangle$ is valid iff for every occurrence sequence σ of G_1. T is correct w.r.t σ

5.2 Examples:

5.2.0.a Parallelization:

Figure 3 shows an example of CICN rewriting which is correct w.r.t. the firing sequence $\overline{v_0}v_1\overline{v_1}v_2$ leading to the marking M_1 where only the node v_2 has a token. The correctness criterion is $L = \{ef, fe\}$. The mapping ϕ is represented by dashed lines and is given by $\phi = \{(v_5, v_2), ((v_4, \beta), v_2)\}$. After the change is made, the execution will resume in the state given by the marking of G_2.

5.2.0.b Swapping:

Figure 4 shows an example of CICN rewriting which is not correct w.r.t. the same firing sequence as before, assuming we have the same correctness criterion. Any token mapping will violate the correctness criterion.

5.2.0.c Sequentialization:

Using the same correctness criterion as before, we can easily see that the CICN rewriting represented in Figure 5 is not correct under the marking shown.

5.3 Main Result:

We are now ready to state our main result. In essence, it states that for every CICN rewriting, there is an equivalent one, called *synthetic cut-over CICN rewriting*, which is valid.

5.3.1 Theorem:

For every CICN-R $T = \langle G_1, G_2, L\rangle$ there exists a valid CICN-R $\tilde{T} = \langle G_1, \tilde{G}_2, L\rangle$ such that:

- 1- $\tilde{T}$ is valid
- 2- $G_2 \subseteq \tilde{G}_2$
- 3- $L(G_2) = L(\tilde{G}_2)$

1. Algorithm of construction:

   The construction of the new ICN rewriting is done as follows:

- 1.1 Identify *the primary change region* in G_1. This is a *convex* subgraph of G_1, noted R_1, induced by the set A of nodes altered during the rewriting process, such that all its immediate successor nodes are activity nodes. A subgraph H of G is convex if no path between two nodes of H contains a node outside H. Note that a possible candidate for R_1 is the subgraph composed of all possible paths from any node in A to the predecessor of exit, in this case R_1 has only exit as immediate successor. Let C_1 be the set of immediate successor activity nodes of R_1, also called *the lower change region*. Note that G_1 and G_2 coincide outside the change graph R_1 and the new version of R_1, noted R_2. In other words, we can assume that G_1 is composed of two subgraphs (eventually connected to each other) R_1, R_1' and that G_2 is also composed of two subgraphs (eventually connected to each other) R_2, R_1'.
- 1.2 Add R_1 to G_2.
- 1.3 Embed R_1 into G_2 using the following procedure:

  For every activity node x of L_1 do:

  a. Remove all incoming edges of x in G_2 and let H the newly obtained CICN.

  b. For every node y of G_1, for every node z of G_2 such that (y,x) is an edge of G_1, and (z,x) was an edge of G_2, add an or-node to H with y and z as predecessors and x as successor. $\tilde{G}_2$ is the CICN obtained after this step of the procedure. $\tilde{G}_2$ is composed of the three subgraphs R_1, R_1' and R_2.

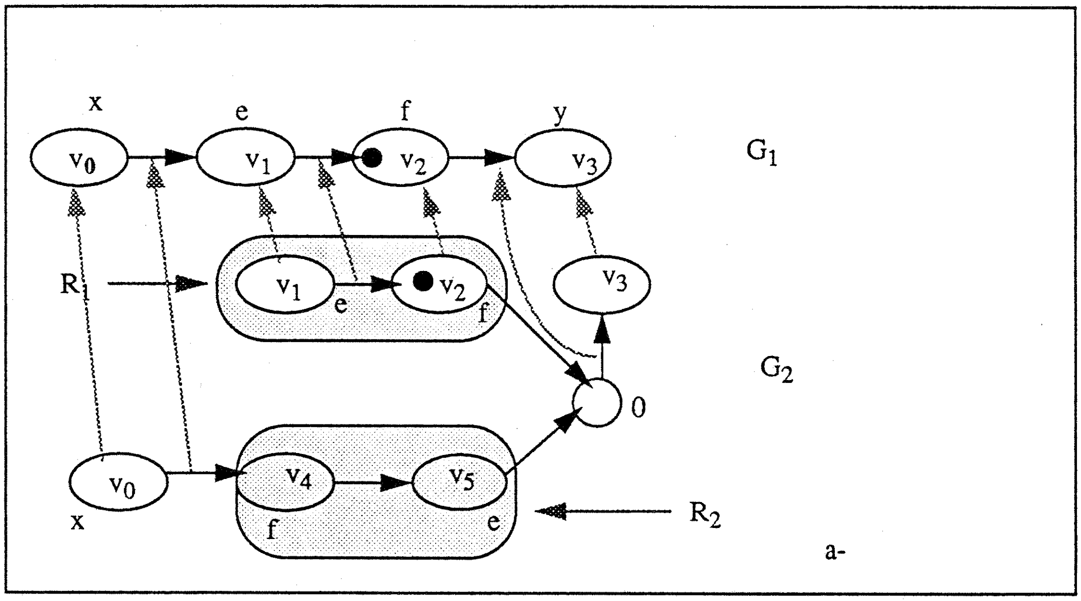
2. Proof of the correctness of the algorithm:

The correctness of the construction is based upon the following observations:

- 2.1 Since R_1 is not reachable (i.e isolated) from the start node in G_1, and G_2, G_1 differ only by the presence of R_1 and R_2, we deduce that $L(G_2) = L(\tilde{G}_2)$
- 2.2 Clearly G_2 is included in $\tilde{G}_2$.
- 2.3 The (token) mapping of definition 5.1.2 will be the same in all states of execution of G1; Φ is not defined in R_2 and coincide with the identity inside R_1 and R_1'. This means that token of G_1 already in progress in R_1 will remain unchanged in $\tilde{G}_2$, and that a token of G_1 in progress outside of R_1 will evolve using the control induced by G_2 in $\tilde{G}_2$ (i.e. in R_2 or R_1'). Whether the correctness criterion is not violated by the new CICN rewriting is straightforward from the construction described earlier. Indeed, note that any token of G_1 outside R_1 will progress in $\tilde{G}_2$ as if it is evolving solely in G_2, and therefore the correctness criterion is not violated. Note also that any token inside R_1, which will exit R_1 after a finite amount of the time, will continue execution in $\tilde{G}_2$ as if it is evolving solely in G_2, and henceforth the correctness criterion is not violated. Finally, note that we should not take into account a token which will never exit R_1 (infinite loops). This is because only computation sequences are considered in the correctness checking mechanism.

5.3.2 Example:

Figure 6-a shows the CICN rewriting given by the construction for the case of swapping, the mapping $\Phi$is represented by dashed lines, and is a partial function. Figure 6-b shows the result of the construction for the case of sequentialization.



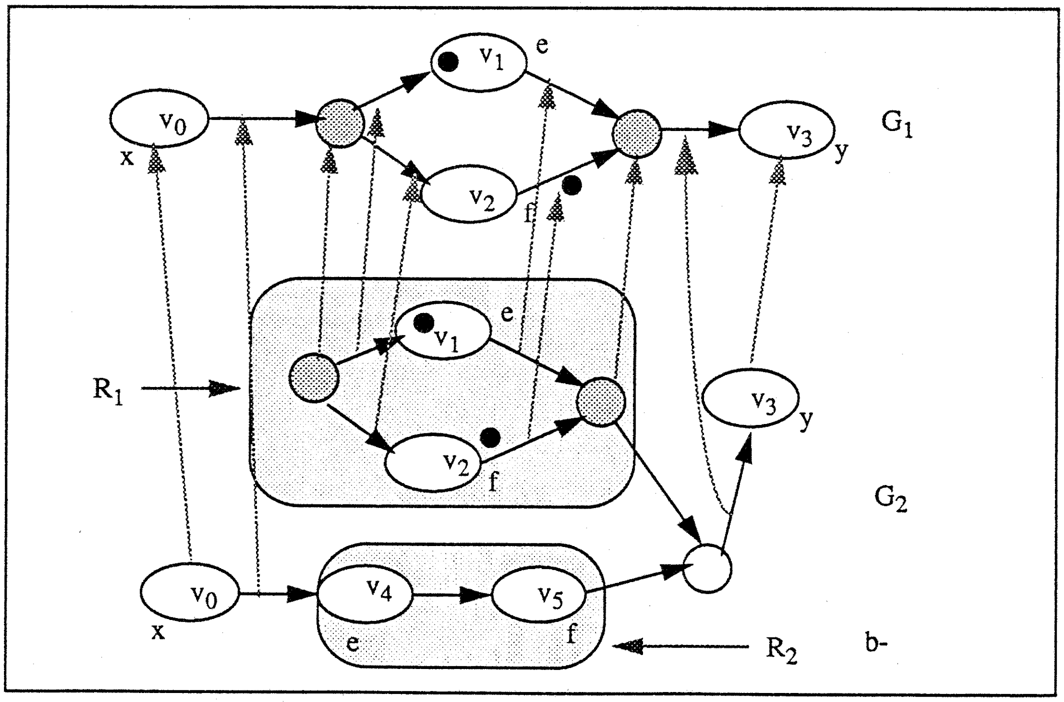


Figure 6: The CICN rewriting obtained by the construction of theorem 5.3.1 for:
a-the swapping
b-the sequentialization

6 Summary and Conclusions:

This paper has presented the ICN as a model of workflow. We have pinpointed the problem of dynamic structural change within workflow systems as an important and challenging problem. Building upon the mathematical formalism of ICN, we have presented a set of mathematical definitions which represent the problem and a general solution which is shown to yield correct results under dynamic change.

There still is a great need to the development of a firmly based theory of ICNs, which will among other things deal primary with excitability and complexity issues such as:

- Can we decide whether or not a CICN Spec is well-defined, that is that $L(G) \subseteq L$?. For what classes of CICNs and correctness criterions this problem is decidable?.
- Can we decide whether or not a CICN rewriting is correct, valid?.
- Under which conditions a CICN-rewriting is always correct, valid?. For instance, it seems that reducing parallelism is not always correct, and that increasing parallelism is always valid.
- Can we find a better class of solutions to the problem of structural dynamic changes in CICNs?. Obviously, the one we have proposed, although correct, is too expensive. Can we take advantage of already available techniques such as graph grammars?
- Is there a better notion of correctness in terms of computability?

Acknowledgments

We would like to thank Andrzej Ehrenfeucht, Gregorz Rozenberg and the members of the CTRG at the university of colorado for many helpful comments and stimulating discussions.

7 Bibliography

Ader, M., Lu, G., "The WooRKS Object Oriented Workflow System," OOPSLA92 Exhibition, booth 712-714, October 19-21, 1992. Developed as part of the ITHACA Research project within the ESPRIT Program.

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

Bair, J. "A Layered Model of Organizations: Communication Processes annd Performance," Journal of Organizational Computing, (2)1, 1991, pp. 187-203.

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

Bull Corporation, FlowPath Functional Specification, Bull S. A., Paris, France, September, 1992.

Cook, C., "Office Streamlining Using the ICN Model and Methodology," Proceedings of the 1980 National Computer Conference. June, 1980.

Croft, W. B. and Lefkowitz, L. S. "Task Support in an Office System," ACM Trans. Office Information Systems 2, 3, July, 1984, pp. 197-212.

De Jong, P. "Structure and Action in Distributed Organizations," Proceedings of ACM COIS'90, April, 1990, pp. 1-10.

Dumas, P. La Methode OSSAD, Les Editions d'Organization, 1991.

yson, Esther,"Workflow," Release 1.0, EDventure Holdings, New York, September, 1992.

Ellis, C. A., "Information Control Nets: A Mathematical Model of Office Information Flow," Proceedings of the 1979 ACM Conference on Simulation, Measurement and Modeling of Computer Systems, August, 1979a, pp. 225-239.

Ellis, C. A. and G. J. Nutt, "Office Information Systems and Computer Science," ACM Computer Surveys, Vol. 12, No. 1 (March, 1980), pp. 27-60.

Ellis, C. "Formal and Informal Models of Office Activity" in Proceedings of the IFIP International Computer Congress, Paris, 1983.

Ellis, C. A., S. J. Gibbs, and G. L. Rein, "Groupware: Some Issues and Experiences," Communications of the ACM, Vol. 34, No. 1 (January, 1991), pp. 38-58.

Grudin, J. "Why CSCW Applications Fail," Proceedings of the CSCW88 Conference, ACM, pp. 85 - 93.

Harel, D., et. al., "STATEMATE: A Working Environment for the Development of Complex Systems," in IEEE Transactions on Software Engineering (16,4) April 1990.

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

Ho, C., Hong, Y. and Kuo, T. "A Society Model for Office Information Systems," ACM Transactions on Office Information Systems, 4, 4, April, 1986, pp. 104-131.

Holt, A., "Diplans: A New Language for the Study and Implementation of Coordination," in ACM Transactions on Office Information Systems (6,2), 1988.

Karbe, B., Ramsperger, N., "Concepts and Implementation of Migrating Office Processes," Verteilte Kunstliche Intelligenz und Kooperatives Arbeiten, 4. Internationaler GI-Kongress Wissensbasierte Systeme, Munchen, Germany, Oct. 1991, pp.136.

Kellner, M., "Software Process Modeling Support for Management Planning and Control," in Proceeding of the First International Conference on the Software Process, IEEE Computer Society, Oct. 1991, pp.8-28.

Kreifelts, T., Licht, U., Seuffert, P. and Woetzel, G. "DOMINO: A System for the Specification and Automation of Cooperative Office Processes," Proc. EUROMICRO'84, edited by Wilson and Myrhaug, 1984, pp. 3-41.

Li, Jianzhong. AMS: A Declarative Formalism for Hierarchical Representation of Procedural Knowledge, PhD Thesis, L'Ecole Nationale Superieure des Telecommunications, Paris, France, December, 1990.

Loucopoulos, P., Katsouli, E., "Modelling Business Rules in an Office Environment" in SIGOIS Bulletin, 13,2 (August, 1992).

Luqi "A Graph Model for Software Evolution," IEEE Transactions on Software Engineering, 16, 8, August 1990.

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.

Poltrock, S., Grudin, J., "Tutorial on Computer Supported Cooperative Work and Groupware," Presented at ACM SIGCHI Conference on Human Factors in Computing Systems, New Orleans, April 27, 1991.

Rein, G., Organization Design Viewed as a Group Proces Using Coordination Technology, PhD Thesis Dissertation, Department of Information Systems, University of Texas at Austin. May 1992.

Sarin, K. S., Abbott, K. R. and McCarthy, D. R. "A Process Model and System for Supporting Collaborative Work," ACM COCS'91, pp. 213-224.

Sirbu, M., Schoichet, S., Kunin, J. S., Hammer, M. and Sutherland, J. "OAM: An Office Analysis Methodology," Behaviour and Information Technology, 3,1, 1984, pp.25-39.

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.

Tsichritzis, D., "Forms Management" Communications of the ACM, 25,7 (July 1982), pp. 453-478.

Woo, C. "SACT: A Tool for Automating Semi-Structured Organizational Communication," Proceedings of ACM COIS'90,April, 1990, pp.89-98.

Zisman, M. D. Representation, Specification, and Automation of Office Procedures, Ph.D. dissertation, Wharton School, University of Penn., 1977.