

## Introduction to Start to Finish (STF): A workflow process router

In the following white paper narrative, an overview of Information Technology Sources product “Start-To-Finish” (STF), a freeware software product that addresses using a software process router to drive workflow. In addition, some motivation and theory of process description and implementation appears, so as to provide some rationale for the design of STF.

### Work

What is work? In most tasks today, work consists of effort toward manufacturing objects, delivering services, or creating and shaping information. In all these cases, information describes what is to be done, by when, and for whom, among other things. Information fills in the ‘what’ the ‘how’ the ‘who’ and ‘when’ of our activity. Information, communicated among us, organizes and directs our activity so as to render productive output we call work. Information, communicated properly directs work, whatever we do. At the end of a task, information communicates the result of a task to managers or customers. Again, information must carry the results of work to others. Even pay for work constitutes communication. These days, we seldom get actual money, but reports of money being deposited, or moved, as ‘pay’.

Information does not just hang out in the Ether alone; it exists in documents. Email, memos, videos, forms, recordings, all constitutes documents. In today’s discontinuous environments, we increasingly receive tasks, perform them, and report on them in formal documents. In this way, documents *contain* work direction, content and reporting. Workflow implementations and management create, direct, deliver, produce and archive documents. Document ‘flow’ is workflow.

### Processes and Content

In performing activities, we make a distinction between the *content* of work, and the *process* of work. If we consider cooking, the recipe concerns process - the step-by-step method of selecting and combining content to achieve a goal - the object of the work. Many computer programs prepare content; the entire spectrum of computer science seems almost tailor-made to create and package content. Very few address processes - the way that people perform activities.

This paper addresses process explicitly: how to do it, how to plan it, how to carry it out, and how to improve it. For our purposes, consider a process a recipe for how to do something. When a process becomes cumbersome and difficult, it usually employs more than a single person, and it involves communication among the people involved. We say that group processes require cooperation, which itself requires communication. Communication exploits expectations. If our expectations differ and do not match our goals, we probably will not achieve them. To become efficient, we must create processes - behaviors - and practice them until we become efficient at them. Creating, documenting, simulating and practicing behaviors defines what process management concerns.

A business consists of interrelated processes. To understand a business, we must understand its processes and improve them. Processes connect through several *functions*. A function describes an isolated activity within a process, although sometimes-complex processes contain functions, which may be processes themselves.

Content, or information somewhat parallels the role of memory in human behaviors. Processes describe transformation or communication of information. We may understand managing memory more completely than we understand managing process, because process management proves difficult when people resist it - which they do in many cases.

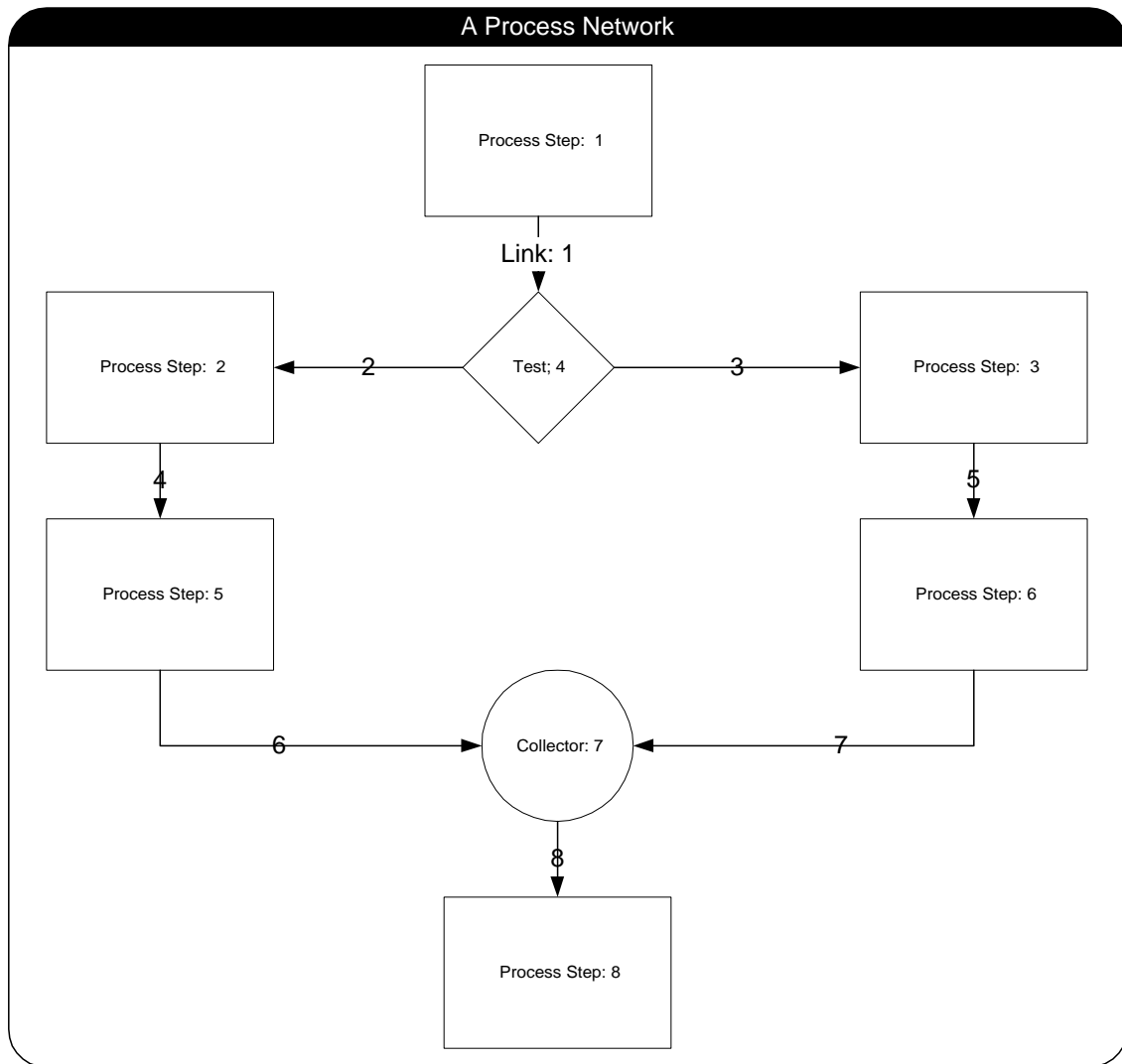
Most organizations contain multiple processes within them. Some processes have higher value than others, and so become the target for improvement and measurement. It is to these processes we look to capture and manage them. Redesigned processes occasionally show dramatic improvement in productivity - hundreds of percent. Well-managed processes complete the picture of what can be done through focusing on, and managing, process.

### Modeling a Process

A process describes a set of sequential and parallel activities - *activities* in time. The temporal nature of process creates a 'one after the other' type of sequence, which lends itself to a list or even a network. A network allows 'circular' processes, where a list or tree does not. A network can be used to describe the most general process. A directed graph is a network, with nodes connected by direction lines, over which items move into nodes. Nodes collect items from the lines. This network may be represented by 'node/link' diagrams, or other types of representations, such as link-lists, which have end points and directions from and to nodes.

A node of the network must have a unique ID (UID), which locates it. It is therefore 'known' by its UID. Node '56', or node "Happy Gilmore" might serve the same purpose, as long as uniqueness is guaranteed. A process then consists of a set of activities, ordered by connecting links. Links, like nodes, *should* have UIDs to locate them in the process, since they actually represent some effort, and a great deal of the difficulty in a process. We want to represent links, their content and their performance, just as we represent nodes, their content and their performance.

A process graph appears next, illustrating UIDs for nodes as well as links.



Note that tests, collectors, etc. represent effort by people, and therefore qualify as nodes. Where an activity takes place a node represents the activity. Node 'activities' represent *transformational* effort: people and machines transform the form or content of information by performing a function on information in some format. Even 'writing a memo' represents transformation of information. Link numbers serve to identify directed segments where work items move, and where information moves from one person to another, or one process frame to another. Link functions frequently overlook communication that takes place at the 'information move' time. This failure to represent the protocols between function nodes accounts for the high incidence of failure of a process description to adequately implement the process itself. Like activity nodes, links take time to design, expense, etc. and these objects collect these attributes of links in a process simulation. Links represent the non-transformation functions within a process. Links implement the communication among both people and machines. We must include the 'informal' paths, which carry essential information, both data and control information, among people, and their machines.

There are multiple ways of representing these 'node/link' graphs. Simple node numbers serve the purpose too, although they become difficult to read when many nodes interconnect. The 'shape' of a process may prove too difficult to create, edit and maintain in the simple list format. Another format is the 'file system' metaphor, popularized by computer file systems.

The Microsoft Explorer represents files in this format. They interconnect in simple ways, but arbitrary interconnectivity cannot be represented without excessive complexity.

A 'work object' may be represented by a 'transaction token' or, simply, a 'token.' The actual work object may consist of a memo, an insurance policy, a speech, or the plans for a city square. In all cases, these objects actually consist of documents. In some cases, activities may involve physical objects, such as a concrete sidewalk. However, since a token 'stands for' or represents all work objects, the physical nature of work objects need not detain us here. The Token *represents* items of work.

Since a token represents a work object, we need to understand how a work object moves through a process. A process represents transformations performed on something, which 'moves through' the process network. In most cases involving knowledge work, the work object actually is a document. Each activity 'transforms' (that is, creates, stores, retrieves, edits, destroys, reads, modifies, transmits, duplicates, etc.) the document. The document 'moves through' the process, one step at a time. At times the document can be in two different activities simultaneously. This occurs when two activity steps can occur simultaneously. Each 'subnet' must be managed separately, when this occurs.

The job of a 'router' or work processor consists in moving tokens through the network of activities. Activities start as the token moves to each node in the network. The Token records the outcome of the activity. Test code performs logic on the token and object's contents to determine the next activity, and the Token moves to the next activity so indicated. This goes on until the work object, or token, traverses the entire process. The process ends when the 'next node' is a terminating one. Only certain activities can terminate a process.

A node or activity then contains:

1. A UID: Unique identifier. This provides a simple way to identify an activity as a Record in a database, or as a class in programming code.
2. A Name - So people can identify it. Names may not be unique, as people can disambiguate names by context.
3. A description - So people can understand what it does, and how it does it.
4. Characteristic - Distinguishing attributes, which might provide clues to understand its relation to other activities.
5. Automation mode - Information system applications, implementations and connections with other applications and systems.
6. An address of a work center, person, or workstation. This allows the node to represent a work step or *activity* in a process
7. A Participant: usually a person or a role (e.g. secretary)
8. An alternate. This is so the time-out can apply, and the token move to the alternate node.
9. Transition information: where the 'flow' goes next. This will usually specify the UID of an activity

## Node Types

Nodes can be of several types;

1. Normal activity or work step in which some activity is performed
2. A 'split' where the work item is sent to two or more places simultaneously. The Split function may include logic, which determines from conditions who to send to and what to send to them.
3. A 'join' where two or more work items must be present to generate further activity beyond the 'join.' The Join function may include logic to describe 'what' to join.
4. A 'dummy' activity, which may only allow time to elapse, but which contains no work activity.

## Attributes of a Node

Activity: An activity represents the work activities, transformation to information and material performed at a place in the process graph. A place represents the 'where' something occurs. Places may only represent a stopping, resting or waiting location, but something describable happens there. The place may only represent a conceptual place, in that the actual location is not known, but nevertheless represents an activity occurring someplace - even if only in cyberspace. An Activity reflects the abilities the user has on the screen. A position defines these capabilities, and so, the appearance and functionality of the software to the person.

Training information: When a new or untrained person takes over an activity or position, they may need training in the work they are to perform. Training may arrive with the activity on the screen. The activity may become 'self describing' as a person is led through the activity. This could bring new people into the positions described in the process with training on the position, as well as its place in the entire process.

Position: A node represents an activity, giving ordinal number to the activity. The activity assigned to a *position* possesses a number by which that activity's position is known. The position of an activity is unique in the process. The Position acts as the Unique ID (UID) of an activity within a process.

Group or Person: A node also represents a person (or, in some cases, the group) who performs work. The person may perform one or more positions, and so, more than one activity. However, the activity usually identifies with a person. Within a group, the person may be one or more people, but eventually, a person receives the work. This may not always be so, but a person should be tagged with the activity if they actually did it. Sometimes a group gets a job, and the work parcels out among people anonymously. Working groups should avoid this practice if possible, as it makes improvement difficult, and tracking of results impossible. Assigning the work to a person can become an activity in itself, and this identifies the person who performs the next activity, the work originally identified as the 'activity'.

When an activity, a position and a person 'come together', the process takes on a defined quality. Persons identified with the activity and position can receive work via the system.

## Link Types

Links represent communication among people, machines and their surrogates. A typical communication link sends a document, or information carrier, from one person to another. The link then represents 'information flow' in the form of a 'document transfer' from one person's 'out box' to another's 'in box.' A link might also implement a short conversation about priority, special conditions, information quirks, and even light chat about something peripheral to the work being done. Links also implement questions about the work, as it progresses within a functional node. For example, when a work document progresses from node 1 to node 2, 2 may question 1 about something 1 did. This requires a communication link, and a corresponding implementation to make this possible, effective and efficient - as well as pleasant to use for the people involved.

Links do not enter into most process designs. Communication within a process appears only as 'arrows', which represent the flow of information between people, departments and functions. Consequently, process designers do not pay much attention to their implementation, except to label them as 'document transfer' or something like that. However, many processes fail due to inadequate or flawed communication. This means that the links lack something they vitally need in order for the process to succeed. Communication may prove more important than the functions themselves, since many functions are obvious to those performing them. The overall flow of information - the 'architecture' - and the communication protocols appear to need design more than telling specialists how to perform activities. And these two items are frequently ignored by most people, as they see them as 'obvious' - which is manifestly not the case.

A link represents a 'connection' to the next *node, position and person* in the process sequence. Because it represents the total link to the next person, the link accounts for the connection of one person to another. Because it connects to the next node, it accounts for the *inter-node protocol* or the information required to perform the next activity from the preceding ones in the process map. So, the link accounts for both the inter-activity protocols and the inter-personal protocols.

In many systems, a common database receives work-related information, from which all in the process can draw. Inter-node protocols simply involve writing into the common database and reading from it. In this case, the protocols describe how a person writes and reads from the database, which describe how a reader finds the data, interprets it and uses it. Similar functions relate to writers. Databases provide the flexibility to represent data in many formats and handles, so users of different needs do not suffer the noise of useless information, and receive the clarity of required data by 'views' of the database tailored to their concerns.

Links of inter-personal communication show how a person 'accesses' another person with whom they have either a request or a message exchange. Interpersonal communication always involves two things: a want and an expectation. These never disappear, even when they appear to. So, we must honor these requirements in our inter-personal architecture.

Personal communication requires that persons can receive shelter from 'noisy' communication at times, and can receive communication when they can best take it. Various channels may be brought to bear here: voice channels, video channels, e-mail, 'chat rooms' or bulletin boards serving a small or large group, as well as other choices. When a process designer implements a link s/he should have observed how people work in these activities to see what the people operating in these positions require in the way of inter-personal communication. Then, the appropriate link design provides it for them. Along the way, the designer might look at productive and anti-productive practices in inter-personal communication, and provide better ways of providing this inter-personal communication links.

Generally, this requires some give-and-take of the designers and users. Many 'back-door' channels among users support productivity, but remain hidden from a designer who only sees the activity of persons abstractly.

Links may be physical, symbolic, emotional, cultural, cognitive and mission-oriented. For example, a mission-oriented link would involve a document that relates to the specific work involved. A law brief, and engineering specification would qualify as a 'mission link'. Transmitting this from one person to another would fill a mission link in their work process.

A Cognitive link might involve a dialog to make a decision, plan a course of action, and derive a logical conclusion. A cognitive link enables two or more persons to perform these actions together. Making a decision might involve some supporting software, which lists alternatives, collects outcomes and records the decision-making process required to make a good decision. This would support the decision making link. There are others, as well.

A cultural link might support common practices, which could generally fall under the 'culture' umbrella: ways in which 'we do business here.' A business culture describes the beliefs, values and attitudes that shape behavior. These links remind or emphasize the nature of these beliefs, values and attitudes. The 'best practice' way of doing a job could be supported by the reference to it, or to enforcement of these practices within a process description, or protocol. In any case, 'best practices' supports the cultural link between persons.

The emotional links between people support the emotional communication that supports a job or process. Ordinarily, we do not include this kind of support, but productive groups do. They celebrate their successes, build internal morale and help each other improve. They remain open with each other for mutual help, communication and training. This means that they can and do praise and critique each other while they support cooperation and positive attitudes to each other's ideas. They maintain their group pride in their accomplishments. These goals are not small ones, and they require 'protocols' which support them. At times, users only require a 'reminding' message to do them. However, most people do not take the time to know when they succeed, or when their group succeeds, and further, do not take the time or effort to carry out the emotional communication that contributes to these goals in their working community. These protocols fall into the management category of behaviors, which a work process should institute, support and maintain. They become part of the process itself.

Symbolic links describe the symbol systems required to support the above links. For example, if conversation is required, then voice transmission should support it. Conversation cannot really occur with most people through email. If 'instant' voice communication is required, perhaps a visual icon a user might select to speak to another person via speakerphone would serve. When these devices implement a more effective, productive and satisfactory process, they should be used. That implies they remain cost-effective. The symbol systems collect the written language, spoken language, interactive image/voice, graphical, spreadsheet, and other symbolic interactions required by the above protocols.

The physical links support symbolic links. Voice communication (telephonic) links, video links, email through the internet or intranet, and other physical links support the symbolic ones. We emphasize that the higher-level protocols are the important ones: the emotional and above level protocols. The lower level ones - those of the symbolic and physical levels, only support the higher-level ones. Many of the symbol systems required imply that a physical link can provide the symbol communication. For example, if personal presence requires people to observe each other speaking, and they cannot appear in the same location, a videoconferencing system may simulate physical proximity. However, videotape may displace these appearances, and among people who know each other well, teleconference provides much of the required 'presence'. Providing process participants with physical links requires study and analysis of the overlying requirements to optimize a design choice.

This hierarchy of linking protocols implements a mission-level protocol, which gets a specific sort of work done. By designing these protocols 'top-down', a designer includes all the necessary communication required for a particular type of process. A competent process design necessarily includes the communication protocols necessary to accomplish the process, along with the activities performed.

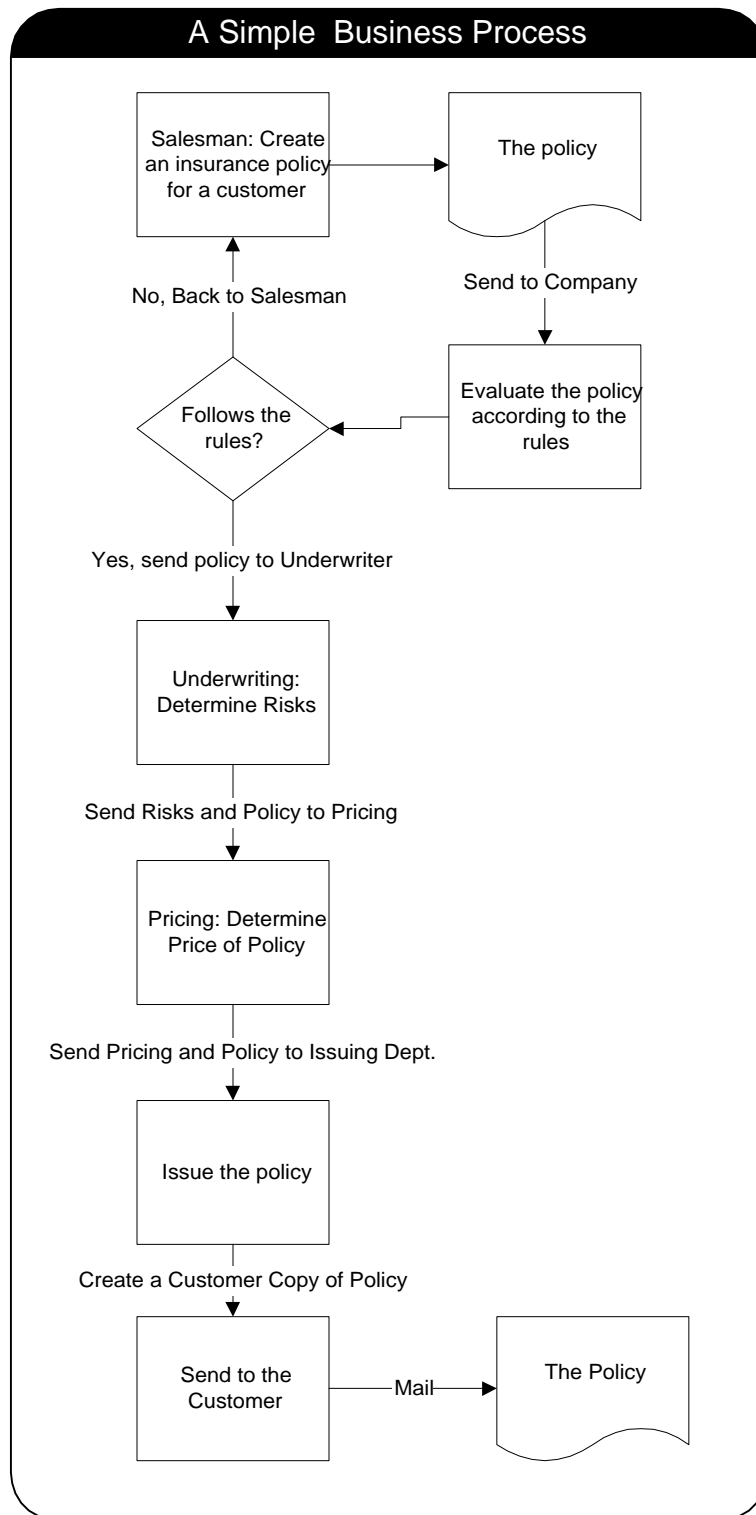
## Human Processes

When a person follows a process, s/he typically either memorizes or refers to a temporal series of steps, which they follow. Steps usually follow an order that must be preserved, though this is not always necessary. Call this the recipe. A recipe transmits the process, as a process, among people. Many other process descriptions also work and may provide more generality, but a recipe keeps the idea simple for us here. Think of a process as a recipe.

In terms of most process 'design', the recipe for a process simplifies information or 'work' flow in order to make it skeletal and understandable. However, the actual communication process around the work place may become more complex than the recipe by orders of magnitude. Some of it may not enter into the work 'system' and some of it may. The inclusion of communication paths (links) among nodes should become a question, with its implementation and even inclusion a major design decision. Again, these questions are usually ignored, and short-changing the communication paths in a work process leaves the architected process impoverished and incomplete. In these cases, impoverished communication yields poor results in the work process itself.

People organize to accomplish complex processes - even when the process remains ill defined or undefined. People seem to muddle through almost any complicated activity, though at times it seems pretty painful and unnecessarily hard. They do this by isolating the activities, which *will* accomplish the overall goal. Each activity then becomes a specialty, sometimes only understood by the people within the activity or function. In the case of major corporations, these functions assume a life of their own, and overall management of these processes becomes literally impossible, because one cannot manage what s/he cannot understand, and these functional 'silos' become impossible to understand. The people within them make that a priority.

If we were to lay out processes people perform, they might look like 'flow charts,' somewhat like the one shown below.



In this case, we see several people involved in the 'flow' of the information, which resulted in issuing an insurance policy to a customer. In addition, we see that for varying conditions, work (that is to say, communication) might take different routes, depending on how any one of these activities turned out.

Companies routinely design and follow processes such as this one. This process might be explained and practiced because it has so few decision points, and so few variations. When the number of variations becomes large, the process becomes less tractable, and people do not follow it well.

A professional football team maintains about 50 plays - their 'processes' - in their playbook at any time. This is because more would complicate what they must practice so much they wouldn't have time to practice any one play sufficiently. They must practice these behaviors, or they will not become expert in them, and identifying them, as 'plays' would have no significance. So they limit their processes to allow them to become expert in them. Needless proliferation reduces the value of each process, or play, reducing the value of practice. On the other hand, too few plays reduce the uncertainty of their opponents, and so allow them to practice defense with certainty. Enough, but not too much complexity typifies good process design. A typical industrial process to prepare a document for a printer, for example, contains almost 1,000 steps, each performed by someone, somewhere.

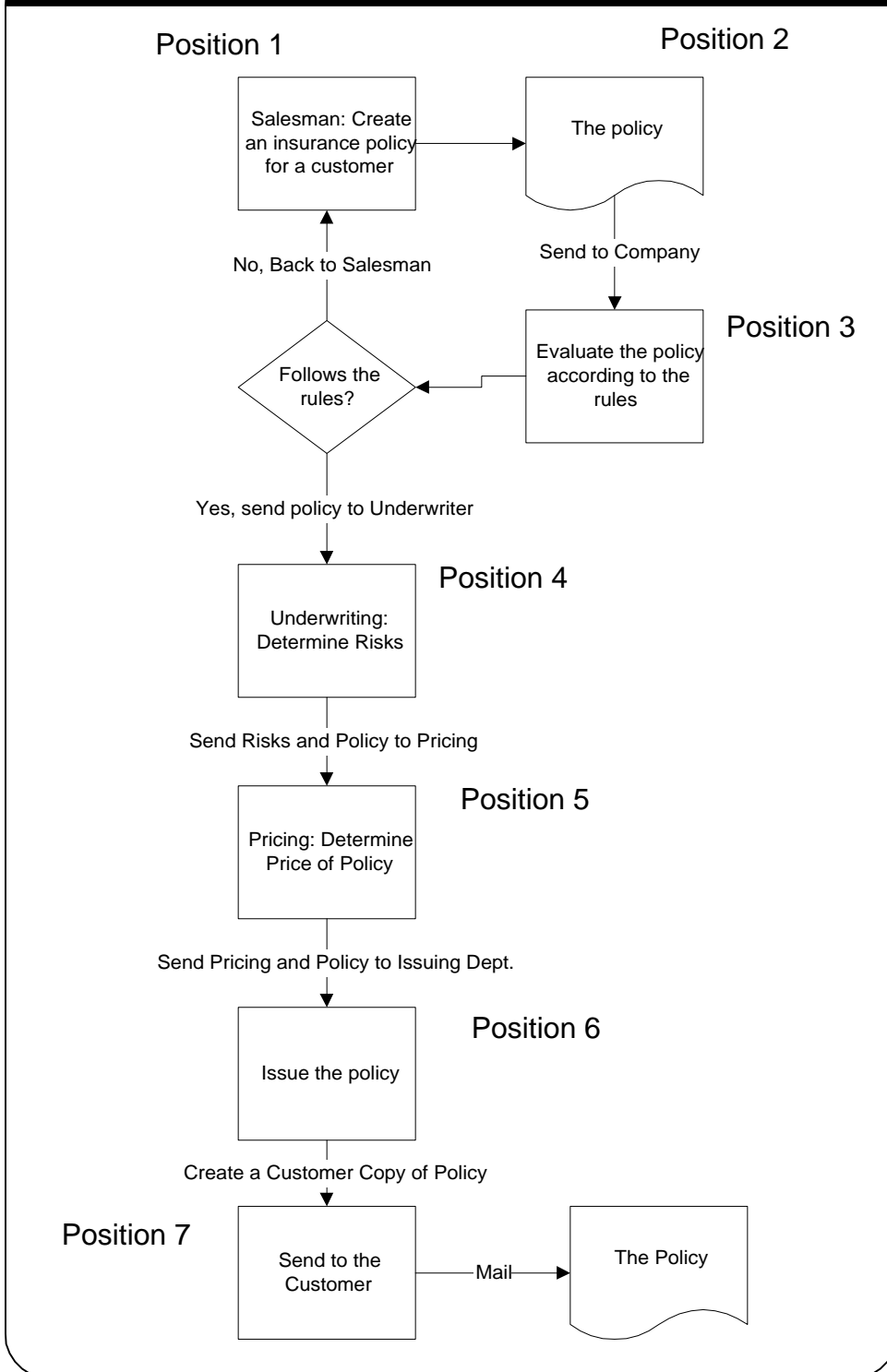
We usually communicate processes with diagrams. Diagrams - graphics - show relationships well, whereas words describe content well, but not relationships. Only very talented writers succeed in creating rich relationship descriptions with words. Processes *are* relationships functioning. The flow charts, such as the one above, typify these diagrams. The relationships involved here are the 'next' temporal relationships of work steps. Temporal before-and-after relationships dominate process diagrams. More subtle relationships may also appear in diagrams. However, diagrams suffer from overcomplication. A too-complex diagram soon becomes useless, as it violates one of the rules of communication - which a communication can break down into manageable pieces, which persons may consult individually.

One of our difficulties today concerns complex processes, and processes with many decision points and 'branches' with occasional activities, which may become numerous. Many processes simply defy understanding by those attempting to follow them. The situation resembles a football team with 3,000 plays, each with small variations. Just as we can capture and maintain coherence of content with information system aids, we would like to similarly capture and maintain coherence of process with these aids. The 'process router,' or process driver serves this purpose.

For this purpose, consider 'work' or 'work items' to be contained in documents. When a person receives a document, this equates to receiving an activity. Documents contain, and carry work.

Consider the process with further definition. In the picture below, we show the same simple process, but with some definition of 'positions' or people who perform a function within a process, and their tasks spelled out in a simple description. This picture appears below.

## A Business Process Further Definition



The task descriptions within the boxes describe the ‘data transformations’ people make. These tasks show what people, or ‘positions’ *do*. The arrows show the communication activities, which also show what people do, but further, show how they *relate* to others within the process.

This step should derive a process description of what the present process is, and how it works. It often embarrasses business people, but this usually yields benefits.

## Introduction to the Work Process Router

Documents contain work information, and communicate it to the participants in the work process. The 'flow' of documentation containing work information implements the 'flow' of work. *Driving workflow boils down to routing documents!* A document router moves a document among and between persons, who may read and edit the document, and change its status. The document may represent a job or unit of work. The persons who act on or because of the document can then perform work described by the document, or described by the person's function. The status of the work or job resides in an auxiliary 'token', which parallels the document in its course.

Below, we describe these functions, the documents to which they apply, and the creation and management of these documents.

The document router implements 'work flow'. In a simple application, a document would be sent to a 'distribution list' of people, each of which would see it in its turn. They might read it, edit it, perform work described in it, and pass it on to the next person. The document might contain the actual work, or only consist of instructions about performing work described there. Each document might contain contents specific to its status and position within its 'distribution list' or milieu.

## Input Database

In most installations, the input for new jobs occurs when users fill out a 'service request' 'problem report', 'Purchase Request' or some other request for service or work to be done. The database records all information relevant to that particular job. Information may accumulate as the work progresses. Generally, organizations prefer to work into such a database, and report out of it, rather than to manipulate the workflow process directly. Among other advantages, no additional software need be installed on a user's desktop computer.

The Input Database population with data occurs when a user puts in a "request" for work. This database must contain all the work-related information necessary to route and process the job. A process generally requires a specific database table layout for inputting information. There may be more than one source of information, such as an Address Book for an Email application as well. However, the Database contains names of persons receiving routings in the workflow process, as well as the Process ID, and the document locations or names containing the work information specific to individuals receiving work.

## Driving a Process

An organization might consider a process driver to be the process diagram, driven by the routing software of a process driver. One problem with this lies with participants in a process who cannot be driven. For example, customers call or come into a system randomly; a process 'responds' to them as they arrive, or as they behave in certain ways. In many environments, process participants are not and cannot be driven by inputs, but seem to function as 'random variables' within a process. A process driver must search for inputs from these people - or 'sources' of information - and respond to them communicatively while maintaining some process control. Only a 'search' process can keep alert to process inputs, which trigger behavior of the process engine. This may be organized on a 'polling' or an 'interrupt' basis.

Polling implies that periodically, a search procedure will go out and attempt to find new input. Interrupt processes assume that a new input will stop an on-going process and draw attention to itself, perhaps in an input port. These two concepts of control seem to provide a wide range of behavior. In the sense that an automated process operates on data from databases, we prefer polling for new data.

Generally, new data, or new input appears in a database as a 'new record', which distinguishes itself by some state variable, which usually is set to 'NULL' or some initial status value. When the polling procedure finds the new record, it creates a process segment which can then drive the process.

Occasionally, there is no one with whom a system can communicate. For example, a piece of work might wait until a person decides who and where it will be done. This could conceivably work inside a known process, but the manager might prefer to work in the confines of a database where all problems can be kept together on a screen. Therefore, s/he will prefer not to receive a separate notification of each new problem that arrives, preferring instead to organize them in a different, more succinct way. Then, s/he might prefer to put the choices directly into the database, rather than to respond to a notification or a communication vehicle.

By following this logic, we see that a process system must respond to participants' responses using the mechanisms built into the communication system, or to databases and their surrogates, which lie in the environment, presenting ever-changing data to the system itself. Each situation, data source or input to the system seems to imply a procedure that uniquely understands that input, the relevant conditions under which it assumes importance, and the process to bring it into the system. For this reason, programming seems inevitable for specific cases of information input. Of course, we would like to 'regularize' input, taking rules from a database to deal with each situation. This doubtless can be done, but it remains complex to program the database to deal with data sources.

### Input Function

For every specific database, an input function must be written to take the names, document locations, and process ID and other information, and populate the Job tables for the specific job. Job tables generally refer to Tokens, Distribution Lists and Distribution Units. These have due dates and other relevant data, which applies to some processes, and not to others.

### States

A state reflects the status of a job in the Input and Reporting database. The status describes what the database 'knows' about the job. It is written by both the User Input and the Routing processes.

**Initial** - A state for jobs, which have been put into the database, but for which routing has not begun.

**Routed** - A state for jobs in the process of being routed.

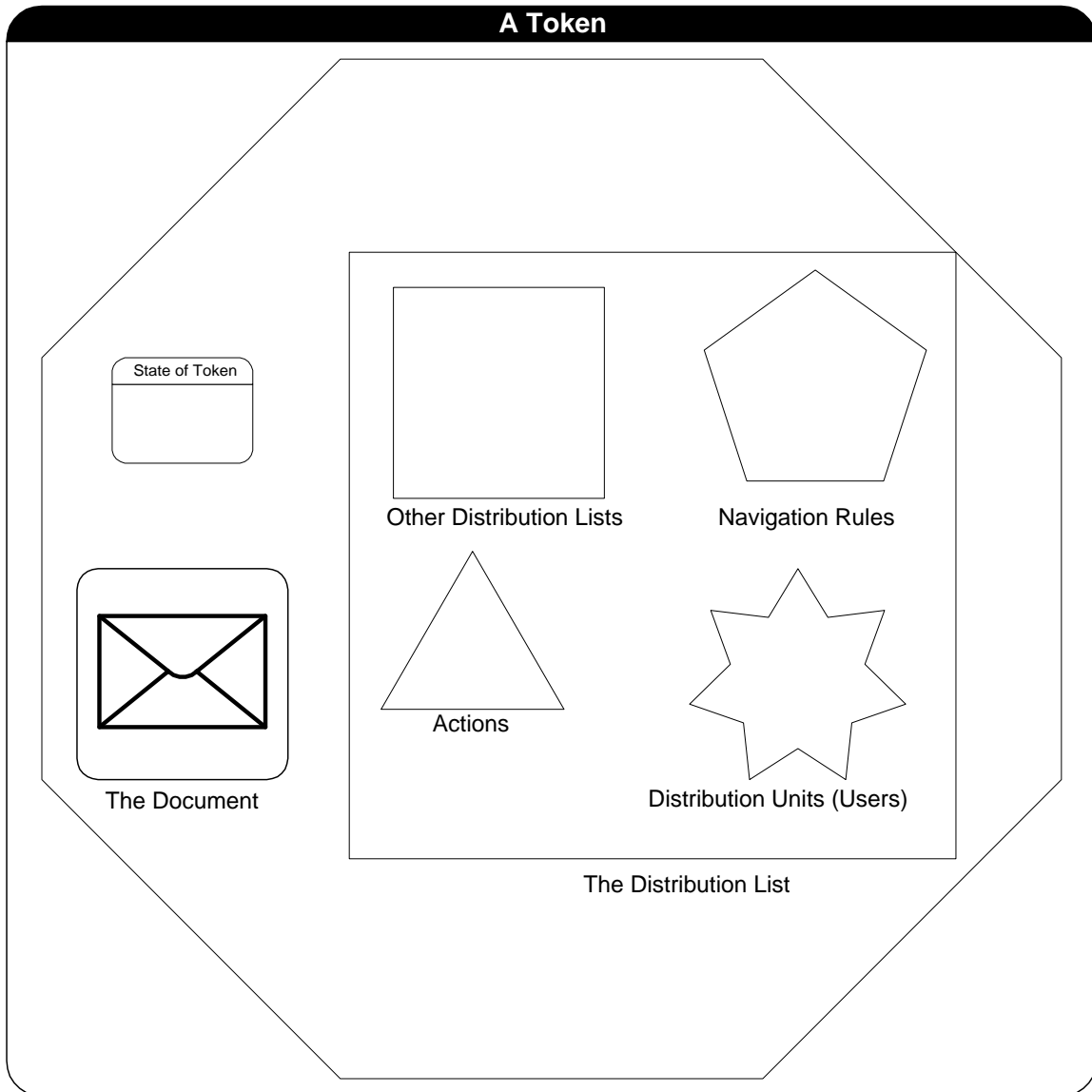
**Approved or Satisfied** - A state for jobs, which have been completed 'normally' and satisfactorily

**Rejected or Unsatisfied** - A state for jobs which have been stopped due to an 'abnormal' or negative determination on the part of a participant. This occurs, for example, in an approval process, when one of the approvers does not approve (rejects) a request. It also occurs when a process terminates due to breakdown, failure or some other reason.

**Unknown** - A state used to show that for some reason the process has suffered an error either within itself or for other reasons, and the software can only report that the result was 'unknown'. Reporting should capture these jobs and either restart them after correcting the error condition, or delete them. Errors in processing generally lead to 'unknown' status.

## Tokens

The 'fiction' created by the software looks like the following diagram. A Token represents the process execution - the particular piece of work to be 'distributed.' The Token contains pointers to the document, and the distribution list that describes the work process. How the lists work will be explained below. This section merely shows how each of these objects relates to the others. The diagram is as follows.



The Token 'contains' the document (through the link to it), the Distribution List, which may contain sublists, and those to whom the tokens are 'delivered' (put in to their InBox), Navigation rules, describing how routing occurs, actions to be taken, and status of the Token itself. Tokens collect all the objects required to perform workflow, and envelop them. Distribution Lists collect and manage connectivity, relating users involved in a workflow process, and the actions to take on work objects at each node. Nodes represent 'where' work is done, or navigation occurs, or both. Navigation rules describe the conditional decisions that determine which path routing takes after an event at a node. Action functions describe 'what' is to be done at each node. The Document provides work content for the activity at a node, and it carries work content away to the next node.

Users may freely modify documents, unless security prevents them doing so. The Token operates like a state machine, with a number of states. Progress through a process follows the transitions described in the Token's 'finite state machine.' As the document flows through the work process, the Token's state, and that of the users' changes, the Token manages the navigation of the Token to affect the process' goals.

A Token implements a particular process. A new process requires a new Token, with its accompanying distribution lists, distribution units, indirectly pointing to concrete persons or directly pointing to them. When a Job enters the system, it identifies the Process to follow. This Process ID is used by the Job creation function to take the Process Template from the Template tables to implement the new job. Job creation then consists of reading the input concrete data with which to fill out the particular persons and documents to which the job refers. This usually comes from a "Job database" with a known structure. In this way, users can input jobs using Internet browser functionality, and forms, which take information structured to describe various processes.

We will describe these below.

The following sections detail the various components and their functions. A complete user interface will be specified. However, the user interface can take many forms. Two of these will be covered, the direct 'InBox' and management interface, and an Email and Report Generator interface to management. Work process management does not have to directly be accessed by users. Email can deliver all information, and bring it back into the process envelope.

#### Token Status - States

Processes maintain an internal status, kept in the Token. Theoretically, status only relates to the process, but may contain information relating to other things as well: special circumstances, customer, and notes. A person initiating a work process must have the capability of querying the status of the process. Customers may require this capability as well. Dates received, initiated, status changed, action taken, and so forth must be retained. Results of action functions, and notes on actions must also be kept and retained into archival status. Additional states may be required, depending on the process and its implementation.

Public Const gconstTokenStatusOutOfService As Integer = 0

Public Const gconstTokenStatusCompleted As Integer = 4

Public Const gconstTokenStatusInactive As Integer = 1

Public Const gconstTokenStatusSubmitted As Integer = 2

Public Const gconstTokenStatusRejected As Integer = 3

Public Const gconstTokenStatusHold As Integer = 5

Public Const gconstTokenStatusSuspended As Integer = 6

Public Const gconstTokenStatusReturned As Integer = 7

Public Const gconstTokenStatusCompleteNotified As Integer = 8

## Routing and Routing Rules

"Routing" refers to a programmatic movement - or apparent movement - of a document among persons.

Routing may distribute a document to persons sequentially, or in parallel, or in combinations of these. A job or work originator typically creates a 'distribution list' of people or functions to which a document (i.e. a 'piece of work') is sent. However, others may add to the distribution list, and include the 'action function' each of these persons is to perform to 'satisfy' the activity. For example, one person might add a section to a document, while the next may merely approve the addition. Persons may be added to the list preceding the person adding the addition, or following the person adding the new person. Each routing would result in a different path for the document - following the modified list.

### Serial Routing

A serial routing describes a path of persons, or functional equivalents, consisting of sequential nodes, each of which is a person. A serial route would distribute a document to the first 'node' first. The first 'node' or person then performs something to the document's state - perhaps only acknowledging that they 'got it.' This state change triggers the movement of the document to the second person, and so on. State changes (which will be described below) trigger routing movement.

### Parallel Routing

A parallel route describes the process of sending a document to a number of people or functions (one or more) at the same time. These people then can view and operate on the document simultaneously. The document rules determine what state changes subsequently move the document to the next 'node.' The parallel node consists of multiple people or functions. Many behaviors might govern the status of a document in a parallel node.

### Visibility of Routings

Routings should be viewable by those constructing the routing paths, and those allowed to see them. Nodes - persons - must provide viewing of their names, job function, organization, building or location, phone and email address, as well as the snail mail address. More may be added.

### Storage of Routings

Routings may be stored and retrieved. They may be reused, and attached simply to a piece or work - a document.

### Routing Rules and Navigation

Rules determine how a document routes from a node to another node. The rules which govern the routing may consist of information within a token, and executed directly by a user - such as a 'send' or 'function complete' button. Rules might use data within the document, or within other control structures, or the user interface itself. Of course, a document may serve as a conduit to underlying databases, where any data may serve to route a document. Rules determine what happens next, given the information available to the rule executors at the rule 'firing' time. Generally, without specific rules, a routing takes a 'normal' route. For example, in a serial list, the document would go the *next* person.

Normally, routing moves from the 'top' of a Distribution List (see diagram below), to the 'bottom' as seen by a user. A Token, representing a document, or piece of work, contains the master Distribution List. Routing commences by taking the 'topmost' name first. If the topmost name is a *parallel* name, all succeeding parallel names are used, and the document 'sent' to all of them simultaneously.

In the absence of special rules, when the first person or group satisfies the action function, the document is 'sent' to the next name, or group of parallel names. Status is carried in the Distribution Unit (DU), as 'state.' When the state of a DU is changed to 'delivered', the DU can see it in the InBox. Otherwise, the document cannot be seen by that user, except in the 'accessed' state.

Rules govern the movement of the document. Documents may move back into the distribution list, for example, and be returned to the originator, or terminated. Theoretically, any possible movement may occur.

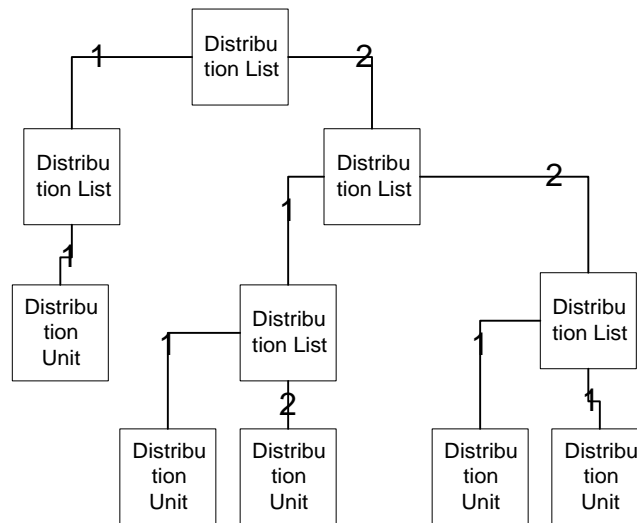
Some persons receiving the document may have no function at all, and not influence the process of the document. These people have a NULL action function. They merely receive the document for information only.

### Nodes - or Persons and Functions

Nodes describe a person or function, which stands for one or more people who fill a function. Since a function may describe a job which persons do not fill every day, a function 'stands for' a person-in-a-role. This may also describe other locations which persons relate to in some way. We assume that a function ultimately terminates on one or more persons or automata who can 'read' the document and respond to it.

A node - a person or function - must be discernable, and traceable to a person at any time. A person may appear in a process multiple times.

A node - a person or persons - are identified and coded as 'Distribution Units' or "DUs". DUs attach to Distribution Lists - "DLs". A DL contains other DLs and DUs. The 'tree' of DLs and DUs contains the routing information required to perform the process.



### A Distribution List

The diagram above shows a typical network of DLs and DUs. Numbers on the links show the sequence of notification, and so of the process flow. Note that the two DUs on the lower right both show a "1", indicating that they are to receive the document simultaneously, i.e. in 'parallel.'

Notice that a DL may contain serial or parallel DLs. In the case of parallel DLs, they execute (i.e. route) simultaneously, and the lists inside them treated as they are marked: serial or parallel. Thus, circulation may become complex, depending on the routing rules inside the lists. Also note a limitation on interconnection. A DL may contain *either DLs or DUs, but not both*. This does not limit the generality of the structure, only the assembly rules.

Routing in a case such as the above depends on the rules at the node representing the parallel split. When the rules execute, the DUs under all other lists under that node may be marked according to the rules at the node. A node's action rules potentially affect all DLs and DUs under that node.

A Distribution List acts as a State Machine. The States are as follows:

- Status Inactive
- Status Active
- Status Satisfied
- Status No Change
- Status Rejected

Inactive states show the list to be inactivated yet. It has not been used. Generally, this is an initial state for a DL. When it is in process, it moves to the Active state. When a satisfactory or normal conclusion has occurred, state changes to Satisfied. All these states may have state reported 'upward' during processing. The No Change state does not represent a state, but reporting on a state change. The Rejected state identifies a DL in which the action was not normal, but the process was stopped as the work output was Rejected. This occurs more naturally in approval cycles, where a document was not approved, but rejected, for example.

### The Distribution Unit

A DU represents a person or group receiving documents or work. It has a state, which records the action taken by the user on the document. States allow the Token to manage distribution, and record the history of the process as it executed. States are the following:

- Inactive - User has not yet 'seen' the token.
- Delivered - The Token has been placed in the user's InBox.
- Accessed - prevents 'time-out'. *Accessed* means the user has looked at the document. The user 'knows' its there, and the user is there. The document will not time-out.
- Null - No action ever taken. This can occur in multiple ways. If a parallel group receives a document, and the action function specifies only 1 of the 5 people must 'approve' the document, and one does, the other 4 get a "Null" status in their DUs for this Token. This removes the document's appearance in their InBox.
- Rejected - not approved. Returned to Originator. This occurs when the user 'rejects' the document, and will not process it further.

- Done/Approved/Completed - Normally passes to next person on the DL. This action normally communicates a successful action of a standard kind by the user. This state may have multiple sub-states to indicate shades of actions.
- Time-Out - after a specified time in the 'Delivered' state, an 'alternative' user receives the Token, and the user's DU is marked with this status. After two time-outs, the document is returned to the originator (this is the default handling).
- Returned - to Originator. Presumably, this allows a user to have a change made, and possibly resubmitted.

### Primary and Alternate Nodes

A primary node receives the document, and may have an active role. An alternate node may be specified, but may not receive the document unless the state of the primary DU meets certain criteria. For example, a person receiving a piece of work, but who cannot due to a vacation will have an alternate specified to receive it.

A person may 'designate' another person to 'cover'. This is a choice of the node.

### Time and "Time-Outs"

The Token operates as a finite state machine. Time influences some states. For example, when a Token lies in an InBox for longer than a specified interval (the time-out interval), the Token "times out" and routes the Token to a specified alternate.

The record of process function also records time-stamps on each event and state change. The time-stamp records the exact time of the process step for archival and study purposes.

A node may receive a document, and not respond to it for a period of time. The time within which a node must respond so as not to trigger an automatic response can be set as it's 'time-out' period. After then, an alternative action or rule may trigger. For example, if you do not return my telephone call before 12 Noon, I'll ask someone else to go to lunch.

### Configurable Routing Rules

Rules must be configurable to facilitate routing changes and procedures.

### Action Functions

An *action function* describes 'what is to be done.' For example, a plumber receiving a document describing a dripping faucet, which he is to fix, describes the 'document' and the 'action function' - "fix the faucet". When he does this, he might merely click the 'done' button on the document, which indicates he did the piece of work requested. Actions relate to a 'node.' Action functions query a database, and/or the document, which passes work to the node. Usually, the document forms or contains a link to databases where decision data may reside.

Action functions may become quite complex, and take on the character of program segments that transform the state of the world as we know it. We will implement simple action functions initially, which may be implemented by 'done buttons' and acknowledged by users at the document screen. We do this on server pages on the Internet when we enter information and 'submit' it, for example.

More complex action functions may become detectable by the information system, and so integrate into the work processor itself. This type of software might detect an event, which signals the status of an activity, and then act on this status.

Action functions, and their results form a permanent record archive of the process.

Action functions may become quite complex. For example, 'batches' of simple actions may be satisfied with a single action. For example, an entire day's 'orders' might be 'approved' in one keystroke.

## Security

The database must provide security consistent with controlled work functions. Privileges and rights must be available to the people who manage, control, and use the Routing functions as well as action functions, and even the document content read and modified by system users.

When documents and other information is sent to users, it is encrypted, so as to add a layer of security to information sent over the internet, and local area networks. This includes documents and other secure information in the access strings and control interfaces.

## Audits

Auditors must audit the creation, modification, and work steps later. This indicates that a log of these transactions be kept.

## Archiving

Processes, down to individual pieces of work traveling through a process, must archive itself as a last step in the process.

## Notification and Distribution

Notification of a user that a document, or work item has arrived at the user's 'node,' requires a way to alert the user to that fact. Generally, a user has an 'InBox' which contains 'documents' which await action by the specific user or user group. (The InBox does not contain the physical document, but only a link to that document. Representing the document with a Token does this. The Token is a software object containing enough information to represent a document to the user and the system. The document does not physically move, but remains in document storage. The InBox only makes the document 'appear' to arrive at the user's screen.) The 'InBox' consists of an application residing on a user's desktop.

The distribution function reads a Table in the Routing database called "WorkLists" which records work items, which receives immediate distribution. The WorkLists table also receives results of a user's work. The Routing functions put and receive information to users through this table. With the InBox application a user may directly read this table" entries for him or herself. Using Email, the Email program interfaces to the WorkLists table through an interfacing function, which reads the table and sends Email, and receives Email and writes results back into the WorkLists table. This table carries all information pertinent to the individual's work item: descriptions, document pointers, status, due dates and so on.

Groups of documents awaiting attention form the 'action list.'

There may be a 'nag option' to remind a user of a near-due or past due activity.

The application may notify requestors of status changes. For example, when a document has action taken on it by a user, and its status changes, the application might notify an originator of this fact.

Public Const gconstWorkStatusInactive As Integer = 1

Public Const gconstWorkStatusDelivered As Integer = 2

Public Const gconstWorkStatusAccessed As Integer = 3

Public Const gconstWorkStatusSatisfied As Integer = 4

Public Const gconstWorkStatusRejected As Integer = 5

Public Const gconstWorkStatusTimedOut As Integer = 6

Public Const gconstWorkStatusForwarded As Integer = 7

Public Const gconstWorkStatusInfo As Integer = 8

Public Const gconstWorkStatusNull As Integer = 9

## User Interface

User interfaces may be used with the work processor directly, creating distribution lists, adding people, and keeping address lists within it's own structure.

### Direct User Interface

User interfaces create a 'functional specification' or description to a program or system. This part of the specification describes 'what' it does.

The metaphor for pieces of work is the 'Document.' As instructions on a piece of paper describes 'what to do', the Document records instructions, data, etc. concerning a job - the 'metadata' for the job. Documents represent containers for information, and serve to identify a piece of work, because either the document describes what is to be done, or the document *is* what is to be done. In many types of knowledge work, documents comprise the content of work. The 'metadata' contains information about the activity, governing the activity. The 'data' of a job contains information to be transformed within the scope of an activity.

The Distribution List implements the routing and connectivity of the process. A user can create a parallel or serial distribution list. He can attach them to each other to create an arbitrarily complex network of lists. A network of lists may encode the most complex distribution the application allows. Users can edit the network: remove, add or move lists within lists, and create rules for navigation and routing, action functions for specific actions at the nodes, as well as interaction rules for the document.

The Distribution Unit or "DU" represents persons and functional groups to which the system presents documents. Users may attach DUs to DLs, representing 'who' is notified of a document 'arriving' at their InBox. The InBox graphically represents arriving documents, and presents a document management interface for use at a workstation PC. It emulates a mailbox. The software objects underlying the implementation mirror the reality of the situation. The software should function as the objects might if they were physical instead of information objects. In other words, a piece of work consists of orders written on a piece of paper - a document.

This piece of paper circulates among those responsible for the work - in order. This 'order' represents a list of people who must see the paper. As each person does their part, they pass it on to the next person until the activity is complete.

Screens are required to do the following:

1. "Process Design" and management Screen - Allows a user to create, edit and manage the workflow process as a whole. This screen provides management functions for the DL, DU, Document identification and linking, Token creation and submission of the Token as a job. Each of these objects presents its own interface for user editing.
  - Design a process - create a distribution list, and edit it. Create an ID for the process/document. Find the ID, given dates, originators, document Ids/names. Edit a Distribution List. Store and retrieve/reuse these process lists. Represent the process graphically. Generally, this should be a graphical 'tree' or a Windows directory tree (There is support for this in Windows). This should provide a general mechanism to create a process structure, and edit action functions at the nodes, move people to lists, and create an order for document circulation to them. Users may insert a person into a list, and specify an action (see below) - including 'no action' or FYI.
  - Specify alternative routing. This is useful when a Token 'times out' and must be sent elsewhere.
  - Specify a 'time out' for Tokens. After this time, routing goes to an alternate DU.
  - Create Action Functions. Edit them. Action functions describe what event triggers a routing decision within a specific node - or within a range of nodes.
  - Designate an existing document as a 'work item.' Documents of any type may be indicated. This designation 'captures' the path name to a document. This name then is stored in the Token, and may be used to retrieve the document. Alternatively, an object name may be used when a CORBA or equivalent object broker can find the document.
  - Submit a document to a process. This starts the process, and notifies the first person of reception of the document.
2. InBox - The user icon and interface to show documents in their action list, and provide document access and action indications.
  - Notify users of items of work (documents) awaiting their attention
  - Interact with the Action List - Through the InBox. Each document may have attachments, etc, all of which may present themselves to the user who selects a document from the Action List.
  - Allow users to 'open' documents by simply clicking on them in their InBox. Document modification uses the document editor native to that document. The document editor must be available to the user's workstation.
  - Execute the Action Functions - Initially this only means to click the 'done' button, but later may include automatic detection of criteria and data indicating status.
  - Delegate to another InBox. This action has the effect of delegating one's power to deal with action lists to another person.
  - Specify alternative routing.

- Keep a list of 'subtasks' as a set of deliverables in the work item.
  - Work through the internet/browser.
  - If possible, integrate into Outlook.
3. Reporting Interface - to use in managing active process and documents
- Query for status of a process/document
  - Query for document/work types
  - Archive, and search the archive using process ID, document type/ID, Originator etc. The archiving database should allow any parameter in the Token to be used to query concerning any Token or Token type.
  - Query for in-process objects: Tokens, types, etc. - same as archive search on parameters and groupings.
  - Provide for reports on how departments did on their work: days to handle requests, how many, etc. (LGH)

#### E-Mail (Outlook) - Internet User Interface

Users may implement the Work processor as an automated work process implementation. This implementation may assume the form of an Internet-based application, or as an Intranet application, using the functions of Exchange Server, for example, for name and address entry and validation.

For example, when a Service request, implemented as an Inter/Intranet service is submitted by a user, the result, through an Active Server Page, gets put into a "Service Database". The Work processor reads this database periodically, to access new work items. When it finds a new Service Request in the database, it creates a work process job (a Token, a Distribution, and some Distribution Units or people to whom work is to be sent). These are taken from the Service Database, or from The Service Database and the Address Book of the organization. The key to this automation lies in the creation of an automated work process, which the Database linking process uses to create the job, distribution lists, and Distribution information (e.g. email addresses and names) for those to whom work will flow.

In installations with Exchange Server, the Address Book may be accessed and used for both validation and inputting of these names.

In 'approval' situations, these lists tend to be simple, linear, sequential lists of approvers, taken from the hierarchy of the requestor's organization, and extended to cover various dollar amounts of the request under consideration. The appropriate list is then constructed. The first approver then receives an email, with 'voting buttons' (e.g. two buttons with 'Approve' and 'Reject' on them). When one of these buttons is pushed, the word "Approve:" (or "Reject:") appears first in the Subject line of the return message. When some message is entered by the approver, it, along with the approval, is entered into the Service Database.

These approvals record the status of the Service Request. Reports or queries to these databases may be used to track the processes of work. In this way, no part of the work processor is actually touched by users. Only system administrators keep the software running, and the databases up. The work processor can automatically clean its own database after each job completes, and originators notified of approval or rejection.

In installations such as this one, there is no user interface. The using organization relies on emails and its database reporting and management for information on work processes and their status.

Alternatively, the two databases may be used together. The Management component of the Work Processor shows graphically how the work process is progressing. This may be used as a dynamic display of work status.

## Overall Token (Job) Management

Tokens are finite-state objects. When an event occurs, states change. When states change, defined actions occur. The programmed actions occurring on state changes drive the process. Users affect state changes directly in Tokens and DUs.

When a state change occurs, such as that which 'uncovers' a Token to a user (in a DU), the DU state is set to 'Delivered.' That user's InBox then displays that document. This occurs by InBox objects inspecting Tokens to detect those for which the InBox must display contents. This mechanism consists of an object linking InBox to Token. This will be called the InBoxToken link object. It maintains double-entry pointers to Tokens and InBoxes. The pointers identify documents. These pointers are managed by Tokens. DUs know their InBox IDs. When a DU state changes, its Token link then creates another link to the DU's InBox.

Tokens are serialized and kept in a relational database. They are not needed, except when a user accesses a document, or when a time-out occurs. Tokens reach an 'end of life' when their state is set to 'Complete', in which case they archive themselves, and remove themselves from the Token store in the database.

## Token Management Logic

Token management begins at the WorkList level, looking for status of 'complete' or a time-out indication from the WorkList entry level. This level is the InBox level, with which users interface. It is in the WorkList that a user will make status changes, and other changes allowable. This level reflects what a user wants to do. When the WorkList record is examined on a polling basis, two events can trigger further processing:

1. The status is 'Complete'
2. Time-out is indicated because the time-out rule specifies a time-out before the present time.

When a Status change in the WorkList is detected, the 'Active DU' record is retrieved, the active DL, within which the DU resides, and the Token which manages the overall work process is retrieved. The DL is queried for the 'complete' conditions, and, if they are met, the DL is marked 'Complete', and the next DL queried for its existence. If there is one, the next DL is activated, the DU placed in the WorkList, the presently Active DU deleted, and the new one entered. In other words, the next user gets the document through the WorkList entry, and the old one deleted.

In case of a time-out, the AlternateDU table is scanned for an alternate, which is substituted for the present one. After two time-outs, the document is sent to the Originator (a DU).

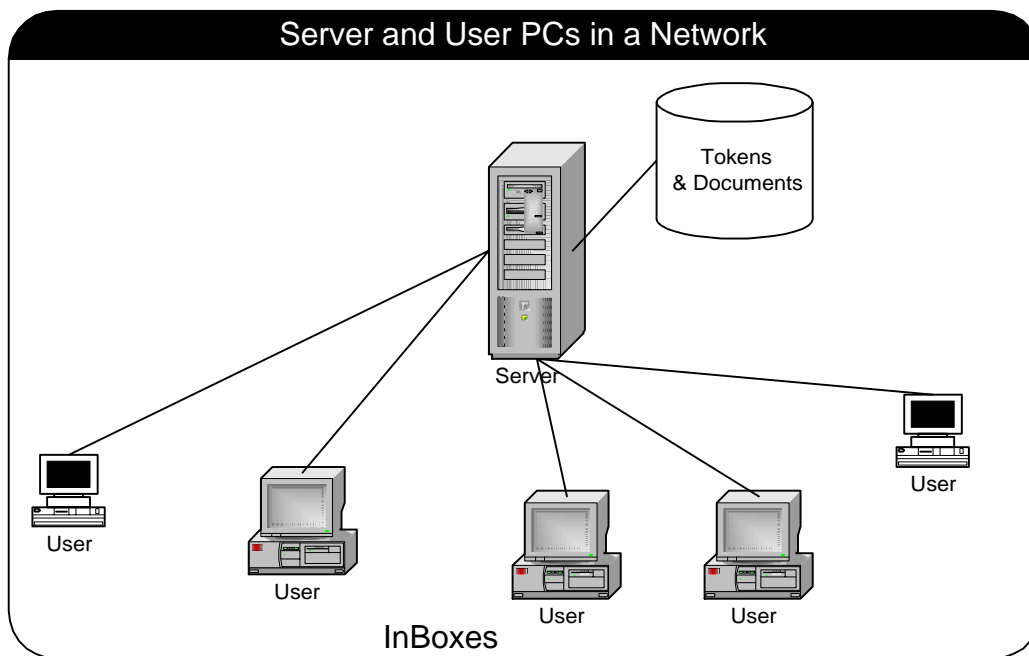
## Physical Configuration of Work Flow Manager

A group of people use the Router, and so access to Tokens must be universal. A central server with a database would keep the Tokens, and allow network access to Tokens. Tokens would exist on the server, and the client PCs would use the Server's processes to access and modify Tokens.

The InBox exists on a user's PC. The link between Token and InBox would point to the PC/InBox on the network. Presumably, Tokens could exist in multiple locations without interfering.

The Document store itself might exist anywhere on the network, including users' file systems on their client machines. This probably would not produce a good environment, but...

The physical configuration is shown below, on a typical LAN. Note that Tokens and Documents are here kept on the server, and users' PCs display their InBoxes. InBoxes link with Tokens through the server linking InBoxToken objects, kept in memory for PCs, which are active and have opened their InBoxes.



## Database Map and Description

The Token database table holds overall job information required to route and administer the workflow job. The TokenID is used to identify the job within the workflow software. The declaration is: `Public grsTokens As Recordset`. This means that the g (the recordset is global, with program scope), re (a RecordSet) and Recordset is a type of ADO (ActiveX Data Object) which interfaces databases to program logic.

The Distribution Unit (or DU) database table is meant to function as an Address Book, as in Outlook. It holds the various names, phone numbers, etc. of people who participate in a workflow process. We use the DU table to hold persons who are responsible for various types of problems. So, by reading a problem type, translating in the DU table, we get a person's Email name.

The Distribution List database table, or DL table, holds an object call a 'distribution list', which functions much like a list of people on paper to whom something must be distributed. They function as described above.

### **'Start to Finish" is Free!**

The software product 'STF' is not marketed as a licensed software product. The development of STF came from previous ITSG projects, which required serious custom-built workflow routing. ITSG retained the right to the product and offers it as freeware. 'Start to Finish' has a sophisticated routing engine that can perform simple to complex routings (serial, parallel or serial/parallel combinations) and can scale up to enterprise level (thousands of users). STF has complete client GUI front end screens out of which work can be created, stored in a database, routed, delivered to users inbox, and perform status tracking and management.

STF is configurable and can be molded to each client's unique workflow processes.

Our revenue business model for this product centers on providing professional implementation services, workflow routing consulting, documentation fee and help desk charges.. Contact ITSG for more information.