A Business Process Environment Supporting Collaborative Planning

Keith D Swenson, Robin J Maxwell, Toshikazu Matsumoto,
Bahram Saghari, Kent Irwin
Fujitsu Open Systems Solutions, Inc.
kswenson@ossi.com

A model for collaborative work process and a graphical language to support this model is presented. The model allows for informal flow of communications and flexible access to information along with a formal flow of responsibility. This approach results in a model of a work process in terms of the communications that team members must make in order to coordinate their tasks. The systems is designed with the intent that the planning activity itself is accomplished in a collaborative way, with many people having input to and control over different parts of the plan. This is contrasted to approaches where the work process is modeled according to the work being done and where the process is either fixed or is entered by a single person.

1.0 Introduction

Information technology has proven to be a great aid in automating and accelerating well defined production activities. While computers have become relatively ubiquitous in the office setting, there is a lack of conclusive evidence suggesting that computers have been successful in increasing productivity of an organization at an aggregate level. There are undoubtedly a number of reasons for this "Productivity Paradox".[23][3][31][19]

One very promising approach to increasing organizational productivity through the use of information technology is termed "Business Process Reengineering"[35][5]. Studies have shown that automating an existing manual work process will have a very slight effect on productivity[23]. Instead, if the entire process is examined, and then redesigned to take into account capabilities provided by information technology, phenomenal increases in productivity can be achieved[12]. These increases in productivity typically arise out of reducing the number of individual steps to complete a process[10].

The new process is made out of tasks which are less menial in nature, and the individual is empowered to play a more influential role in the process. Thus the advantages of BPR are not only those of improving organizational productivity, but also as a way to improve individual "quality of work life".

Before a work process can be improved, it must be visualized and understood. A graphical representation of the work process is needed that not only represents the process, but is also a complete enough description of the process that it can be used to program a process support tool. Furthermore, the visual representation must be clear enough that it can be read easily by all members of a team, not just the process programming specialists.

1.1 Regatta Project Goals

The Regatta group was formed in 1991 in order to develop software to support workgroups and to aid in reengineering work processes[32][33][34]. In order to achieve this the Regatta project has set three specific goals:

1. To make software that supports the *coordination* of activities of different members of a group by automating work processes.
2. To make software that allows every individual in a group to gain a better *understanding* of how their group or organization works. It is not sufficient that a work flow tool act as a black box process. An individual needs to understand how the current task fits into the overall picture. Furthermore, the software must aid in the discovery of the process.
3. To make software that supports the *change* of processes. In order to see any real improvement in the work process, the users must be able to modify and fine tune processes.

This work was supported in full by Fujitsu Ltd. and by Fujitsu Open Systems Solutions, Inc.

Author's present address: Fujitsu Open Systems Solutions, Inc., 3055 Orchard Dr., San Jose, CA, 95134, USA

1.2 How Regatta Differs from Other Approaches

There have been a number of other approaches to supporting and improving work processes. The pioneering work was Michael Zisman's SCOOP project in 1977[36][37], Clarence Ellis' OfficeTalk[4] and BDL from IBM[11]. A number of document imaging and processing companies developed work flow technology in the early 80's. These systems use a number of different methodologies for describing the processes, from hard coded compiled language, to rules based routing, to PERT chart style graphical means.

The commonality between all of these systems is the separation of the planner from the user. The typical scenario involves an audit of the way the process is currently done and how it can be improved by interviewing a representative sample of the people involved in the work. Subsequently a programmer implements the process. Finally the process support application is deployed across the organization. Inefficiencies and inadequacies of the process are fed back to the planner and some time later an improved process will be deployed.

Regatta takes a strikingly different approach; process plans can be created and modified by the end user allowing users to experiment and find the best process. It has been shown that interviewing people about what they do will not always yield an accurate description of what they do[38], so Regatta allows groups to find the process through use and experimentation. Instead of a single programmer for the entire process, Regatta processes are composed from pieces supplied by different users, so we call it Collaborative Planning. Rather than force every group in the organization to work in exactly the same way, different individuals or groups are allowed to have different plans to achieve the same goal, thereby allowing each group to find the plan that works best for them. Process plans can be modified even while being enacted, which allows people to start with incomplete plans and to complete them as they go, and also to respond to exceptions as they arise rather than being forced to make conditions for all possible exception up front.

What Regatta offers to process support might be analogous to what spreadsheets offered to financial reports. In the 70's financial reports were provided by MIS. When one wanted a new report, one gave them the specification, they got a programmer to code it up, and a little while later you would begin receiving those reports on a regular basis. The introduction of spreadsheets dramatically changed this because it allowed people to formulate their own reports, to experiment with new combinations, without waiting, without programming in the traditional sense, and in general to have more control over their own reports. A recent report shows that this "empowerment" is important.[2] Regatta allows people to have control over the processes to meet their goals.

To accomplish this the system needs to do more than just allow users to change processes. There are many design considerations to make sure that such changes happen in a controlled way.

1.3 Requirements of the Visual Process Language (VPL)

Ease of Process Definition: Collaborative processes need to be defined in such a way that the average user could change and create process descriptions. Experience with non-programmers suggests the use of a picture oriented approach to describing a process.[30][15]

User Orientation: The process should appear graphically in a way that matches the user's perception of the process, and not necessarily how the system implements it. The choices and the steps taken by the user should be emphasized, while allowing automated actions to be associated where appropriate.

Parallelism: Teams work in parallel; the process description should allow any number of concurrent activities.

Abstraction and Decomposition: Since each person views what is happening in their own way, it is important that the plan they define works at the level that they do. Higher level people use greater generalizations and "manipulate" at a more abstract level, while lower level people are concerned with the details.

Iteration: The process may include iteration. VPL must be able to represent the case where a task, or process, is repeated until it meets some quality control.

Branches: Decisions may be made that effect how the process is to be completed. VPL must clearly represent the choices to be made, and the subsequent paths.

Delegation: A plan must be able to represent delegation well, so that when one person delegates a task to another, it appears as a delegated task, and not as a myriad of detailed things for the other person to perform. Users need to understand not only that the task exists to be performed, but also some idea of how the task came to be assigned to them.

Control of plan: The system must allow the person who is responsible for a particular result to modify that part of the plan, while protecting that part from modification by others.

1.4 Requirements for the Collaboration Model

Ease of Monitoring Process: In addition to being easy to program, one purpose of the graphical representation is to visually represent the current state of a process to help the participants in the process understand what has happened and what may happen next.

Dynamic Modification: Once a process has begun execution, it may be changed due to special circumstances not foreseeable at the time of definition (e.g. exception handling). A plan (the definition of a process) must be modified easily on a case by case basis, without requiring that the process be restarted at the beginning.

Partial Definitions: Requiring that the process be complete to the n-th detail before starting would be a barrier to use in most situations. It is important that the user is able to define part of the plan in order to start the process, and is able to

add to the plan description as the process advances. It has been suggested that it may not be possible to completely define many processes beforehand due to inconsistent micro-theories of how the task should be done whose conflicts do not get resolved until after the process is started.[14]

Individually Tailorable: The model must be designed such that each person or group can supply their own plan template for a particular task or goal. When an individual receives a request, the system must use the appropriate plan template, "inheriting" a template from the group if no individual template exists.

Authority: Since steps in a process are real and potentially lengthy tasks, the system must be clear about the authority of the origination of the tasks. Any user may create a process, but it should not be possible for a user to impersonate another as the requester of a task. To do this, the system should accurately communicate the authority of the originator of a request.

Negotiation: While the system automates the assignment of tasks to people, it is important that users retain the ability to manage their own work load. An assignee too busy or otherwise unable to handle a task may decline the task, and the system must raise a form of an exception. This mechanism should support a rich free-form communication between the requester and the assignee within the context of the process so that the task might be reassigned, or the plan modified so that work can continue. In a complex system such exceptions might be common, and it must be a feature of the system to support the resolution of such conflicts.

Automation: The system must automate process so that workers can concentrate upon their specialization, spend less time on coordination issues, and make fewer mistakes in forwarding.

Open: The system needs to support the use of tools that are specialized to the user's needs. It should not expect all work to be done within the system. The model should anticipate use of external tools and still provide sequencing support of the various tasks that need to be done with those tools.

2.0 The Model

Primarily, the model provides a shared collaboration space, called a **colloquy**, in which to coordinate a set of tasks that are performed to accomplish a specified goal. Much has been said about the benefit of such a shared space for collaboration. [26][25] ConversationBuilder[17][18] has shown that such an approach can be an effective framework for a wide variety of collaborative activities. Like ConversationBuilder, this model provides for active support of collaboration.

A colloquy is composed of a shared data space and a collection of plans. The shared data space can hold documents and other artifacts. Plans are representations of the process.

2.1 Plans

The model takes a task oriented approach. A plan is a network of stages; a stage is the communications needed to coordinate a task or question. A stage is always a request from one person (the requester) to another person (the assignee). In response, the assignee is expected to complete the task by choosing an option, or to start a subplan to automate at a finer level of detail. This paper uses the words task, request, and stage in a fairly interchangeable manner.

A process starts with a request to someone or to some group. An example of a high level task might be the assignment of the task "Create a New Product" from the president of a company to a product development organization. The task may be accepted or declined (more on this later).

The description of the task is free form text; it is not constrained to a set of predefined tasks. A lengthy description of the job may be included. This allows language to be used naturally for the precise description of the task. Although the handling is different from email, it is useful to think of the stage as an email message that is delivered when the stage is activated.

After acceptance, the assignee is then free to specify a plan to be used to accomplish the task. The plan may be created on the fly with the goal to handle that specific task, or may be read in from a plan template that had been created beforehand. The assignee becomes the owner of this plan. Since the assignee is responsible for the completion of the goal, as owner he is the only person allowed to make changes in this plan.

Each of the stages that compose a plan are themselves requests from the owner of the plan, to the assignee to perform a particular task or answer a question. These requests may in turn be accepted or declined, and if accepted may have subplans in order to implement them. The new assignee is the owner of this new level of plan. In this manner the work to be done is decomposed into smaller tasks.

If the task is sufficiently simple that it need not be decomposed into separate subtasks, then it can be handled manually. The options made available in the request appear on the user interface as menu items. Selecting the menu item can cause the completion of the stage, and activation of the next step in the plan.

Plan templates can be automatically invoked in response to a request. As the work is decomposed, the subtasks become more common and more likely to have a plan template already developed for them, witch can be invoked to automatically create the plan. Since a plan template makes its requests for subtasks in a consistent manner, these requests become in a sense "standard" requests, making it

easy for the recipients to have plan templates in place that are automatically invoked.

2.2 Stages

A stage is either **active** or **inactive**. Any number of stages may be active or inactive concurrently. The stage is active at the time that the people are expected to be performing the task, or taking whatever steps are necessary to make the decision.

Each stage is assigned to a **role**. A role is simply a variable which holds a list of names of people or groups. A role is not an attribute of an individual, but rather a relationship between a person (or group) and a particular colloquy. A given person may play one, two, or all of the roles simultaneously in one colloquy, while playing different roles in another colloquy. The role that is assigned to the stage is said to be responsible for the stage.

A stage includes one or more **options**. Each option has a name that can appear in a menu. A longer description is also available to the user through the use of a help facility. Selection of an option will cause the completion and deactivation of the stage, and will usually send an event to activate another stage.

Events are sent from stage to stage. An event can be named any arbitrary string value. The most common event sent is an activate event, which causes the receiving stage to activate if it is inactive. Finally, the last stage in a plan will send an event to an exit node, which will deactivate the entire plan and send an event to the parent stage in order to communicate the results of the subplan to the parent plan.

2.3 Micro-Options

Options can affect the state of a plan; micro-options affect the state of the stage. These imbedded capabilities of a stage simplify plans for the end user. These options are involved with the offering of and acceptance of tasks within the plan. They help facilitate the process of coming to agreement about a particular task or question.

The stage starts out inactive in a state called unstarted. It remains in this state until it receives an activate event, which causes it to become active.

When first activated, a stage is put into the offered state, if the role is not empty, indicating that the task is being offered to an individual or group. If the role is empty, the stage goes to the unassigned state. If it is offered to an individual it will check to see if that individual has configured the system to automatically accept and invoke a plan from a template. Failing this, the stage waits in offered mode.

Upon seeing that a request has been offered, the user may first review the details about the task. If the task or questions is a simple one the user may take an option directly implicitly accepting the stage. If the task will take an extended time, or if the task is to be decomposed into a plan, then the **accept** micro-option will put the stage in the accepted state. This has the effect of committing the user to do the task. An individual accepting a stage that is assigned to a group of people has the effect of removing the rest of the group from the responsibles list, thereby reserving the task to himself.

Alternately, the user may **decline** the task, which primarily removes himself from the role for that task, and subsequently may cause the stage to go into the unassigned state. When a user declines a stage assigned to a list of users that user is removed from the list. Only when everyone has declined the task, and the list is empty, does it go to the unassigned state.

In the unassigned state the stage appears to be the responsibility of the owner of the plan and thereby brings the owner's attention to the stage; in effect sending it back to the requester. The owner may put it back in the offered state by using the reassign micro-option. By alternating between the offered and unassigned states, the owner and the assignee may carry on a dialog in order to iron out any detail before acceptance (see section 2.4 User Messages). During this process, the owner might change the task, or the person to which the stage is assigned.

When the user actually takes an option that deactivates the stage, or the subplan is completed, then the stage is put into the **completed** state. The third inactive state is reserved for when the stage is deactivated by an external event. An approximate state chart for a stage is shown in Figure 1. See [13] for a thorough treatment of state charts.

2.4 User Messages

The visual process language provides the ability to handle rich and complex processes, but it is important to remember that the main design goal is to facilitate communications between the users. Choosing the "Done" option from a menu on an active task will tell others that the task is done, but little else. Constraining the user to this amount of communications would be insufficiently rich and flexible to handle the minor deviations that all real world tasks involve.

Therefore, with every option users may include a message summarizing information the others participating the

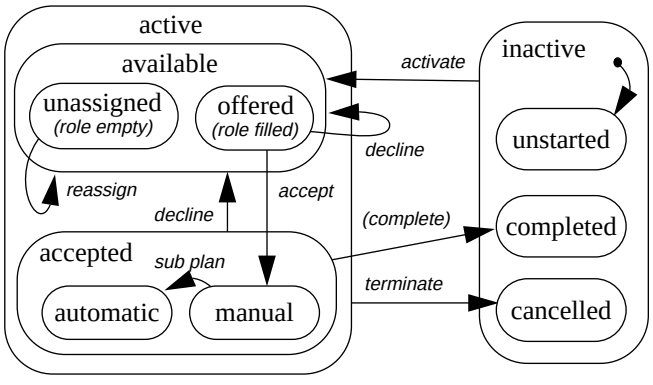

Figure 1. State chart diagram for a stage

colloquy may need to know to complete their tasks. The message can include text, voice, or attached documents of any format. This message is particularly important if a non-standard option is chosen, such as a purchase request rejection. The message explaining why a choice was made, or what needs to be done before a particular option will be taken, is appropriately kept within the context of the colloquy.

Messages may also be associated with micro-options. This is a good way for an assignee to explain why the request has been declined, or to propose a counter offer. The owner may then use the message to accept the counter offer, or perhaps make yet another offer.

The history list is a record of all options and their associated messages and can be browsed at any time by any participant of the colloquy. It is a way for a new participant of the colloquy to catch up on what has happened up to this point. Since date and time are automatically recorded for each option taken, it is easy to determine which tasks are taking the longest and might possibly need some fine tuning of the process.

Finally, since the colloquy functions as a shared work space, any participant may edit the data and/or add a commentary message at any time.

Since history items are made readily available to others, the question of privacy comes up. It is important to remember that events recorded in history are significant events that are needed to coordinate work, such as reaching completion on a certain stage in a project. It is the sort of information routinely communicated at status meetings. Yet some groups like to hold their internal status private, communicating only the aggregate status. To handle this, history is available only to participants of a colloquy, not the general public. If a group participating in a colloquy wishes to keep the handling of a stage private, instead of creating a sub-plan, they will create a child colloquy. The difference is that since the plan will reside in a different colloquy, there is a different list of participants, and the child colloquy is private to them, yet the end result is reported back to the original colloquy. This is analogous to taking a private conversation out of a meeting room, and returning to the meeting with a conclusion.

2.5 Finding Your Tasks

Assigning an active stage to a user is not complete until that user has been made aware that the task exists. In a large organization making heavy use of the system, thousands of active colloquies with hundreds of thousands of individual tasks may exist. Only a very small fraction of those tasks are active at any one time. The system must provide a way to see all of a user's assigned stages quickly and easily. The system provides a window that continually displays all colloquies that contain one or more active stages assigned to the individual. The user is able to distinguish between tasks that have been accepted, and ones that have just been offered.

The active list functions as a to-do list. As new stages become active, they appear on the active list. As options are taken that complete stages, they disappear from the list. When a task is automated by a subplan, it disappears from the list, but possibly stages in the subplan might appear. If a stage is reassigned, it disappears from old assignee's list, and appears on the new assignee's list.

The active list can also be compared to the in-box of a mail system for those companies that use e-mail to coordinate tasks. The e-mail based approach has two disadvantages. First, the e-mail in-box is private and there is no way for the others in the collaborative process to know whether a request has been handled, whether a user has passed the request on to others, or whether the task has started a series of sub-tasks. One of our goals is to let others know this information which is useful for coordination. The other significant disadvantage is that once a message is placed in an in-box it can not be removed except by that user. An email question can not be directed to a group of people and then retracted. Email can not be readdressed the way roles and stages can be reassigned.

Another way to find active tasks is by browsing the parent colloquy / child colloquy relationships. When an action within a colloquy starts another colloquy, the new colloquy is a child-colloquy of the original one. Every colloquy displays a list of all child colloquies, as well as the subset of child colloquies that currently have an active task assigned to the user.

Since stages can be activated in parallel, any number of stages may be active at a given time within a colloquy. When a user views a colloquy, the colloquy is searched for all active stages and are presented as a set of active tasks. If that user is playing a role that is responsible for one or more stages, the options for those stages are presented to the user as menu items. The user may not take any options on stages for which he is not responsible.

2.6 Scripts

If an automated action needs to be performed at a particular point in a process, a **programmed node** is used which includes a user specified script. In the current implementation of the system this automation is provided by an interpreted scripting language called TCL (Tool Command Language)[24].

Options placed on a stage can have programmed side effects. Just after the option is chosen, a user defined script is executed on the client module, and another script is executed on the server, thereby allowing for side effects beyond activating and deactivating stages.

Each stage has two scripts: one that is executed when the stage is activated, and another that is executed when it is deactivated. If a task is supposed to take place with a particular environment setting, it can be set up as the stage is activated. Or a stage might be programmed to send email when it is activated. A stage might be programmed to check a file back into the source maintenance system when it is deactivated.

3.0 Visual Elements

Users use a program, Graphical Planner, to create and edit plans and templates in VPL. The basic elements are nodes which are connected by arcs. There are many classes of nodes: stages, split-stages, AND-node, programmed-node, start-node, etc. Nodes may be either active or inactive. Arcs are placed on nodes in order to program them to respond to particular events, and to send events to other nodes. Arcs represent options that are available to users when that stage is active, as well as the events that are sent as a result of choosing that option.

3.1 Stages

A stage is a node where the process stops for user interaction; other nodes execute without waiting. Stages are represented in VPL by ellipses. The title of the task or question appears in the lower compartment of the ellipse. The assignee role appears in the upper compartment of the ellipse.

A special kind of stage is a split-stage which when activated is duplicated for every person assigned to the stage, so that each person may make their choice independently. A split stage is indicated by a double bordered ellipse.

Small circles on the edge of a stage represent an option for the user. When the stage is active, each option will cause a corresponding item in the options menu to appear to the user who is assigned this task. The letter within the small circle is a first-letter mnemonic for the menu item that is presented to the user.

3.2 Events

An option, if chosen by the user, can send an event to the stage designated by the arrow coming from the option. An event can be specified as any string of characters. The most common event, **activate**, does not require a label.

The option will normally deactivate the stage that it is upon. Alternately, an option may be set so that the event is sent without deactivating the stage, allowing both stages to be active. This is indicated by use of inverse coloration of the option.

An event may be sent when a node of any kind activates or deactivates, without any regard to how the node was activated or deactivated. An event triggered upon activation is an arrow without a circle at the base of it.

When a plan is created, the **start node** is automatically activated. It immediately deactivates triggering the events attached to it.

When an **exit node** is activated, it will send the event specified in the exit node to the parent of the plan. There will be an exit node for every different event that the parent stage is expecting to receive. Any stages remaining active within the plan will be terminated by the exit node, thereby stopping all activity within the plan.

When two or more arrows are pointing into a stage it means that the first activate event from any one of them will activate the stage. A stage that is already active ignores any subsequent activate events it might receive. This OR-like behavior is common for all VPL nodes except for the AND-node (described below) which has been included in the language precisely for this reason.

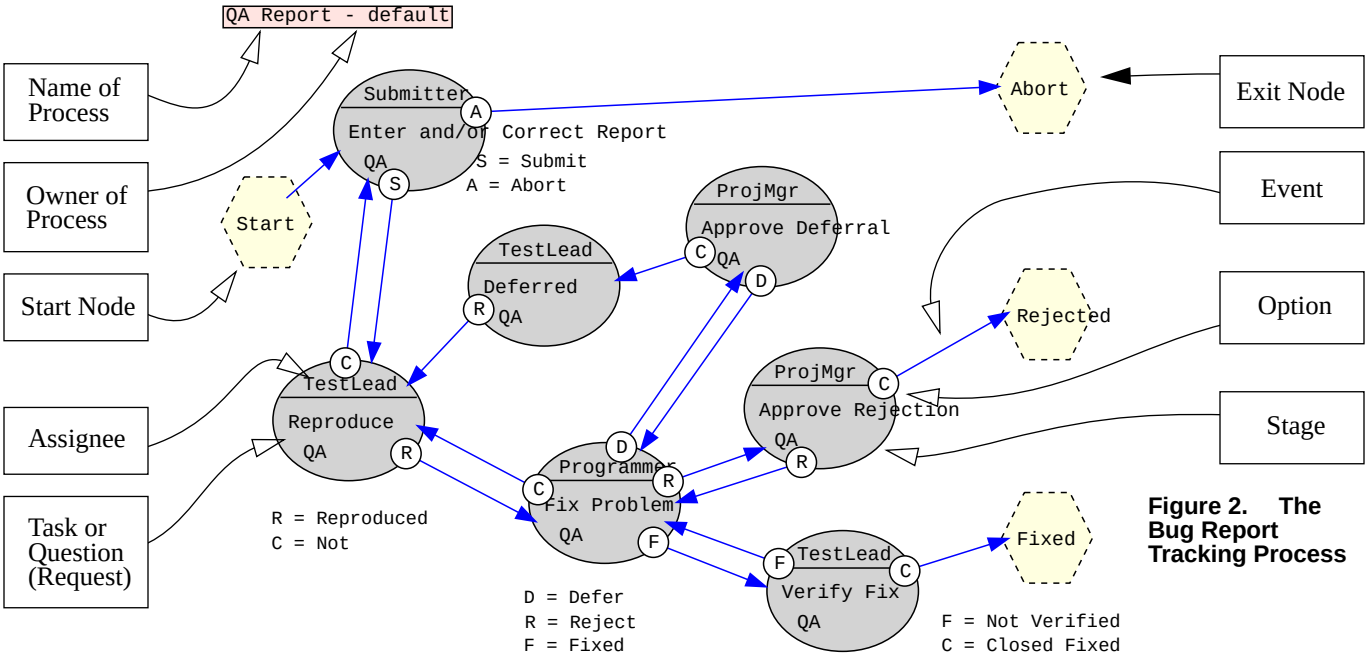

Figure 2. The Bug Report Tracking Process

3.3 Automatic Nodes

The simplest type of automatic node is the *programmed node*. This node is activated by an activate event from any source. It executes a script and immediately deactivates, triggering events that are waiting on deactivation. All automatic nodes may have any number of obligations, signifying that when it is complete, it will send events to any number of stages. This allows any automated node to start parallel tracks in a plan.

In cases where the planner wishes to assure that all of a particular set of events occur before the triggering of the next stage, he must use an *AND-node*. This symbol, a small circle with a plus in it, means that all incoming events must be received before the output event is sent. Common usage would be stages that are triggered when a set of parallel stages are all completed, for example when every person in a group is finished with a particular message or document.

The **timer-node** is a programmed node that has the behavior of waiting for a particular amount of time after activation before deactivating. Any amount of time may be specified from minutes up to weeks or months. The purpose is to allow for reminder style events that appear after a specified time.

In order to program the process so that it activates different task on different conditions, a case style **branch node** is provided. It consists of a set of conditions. The conditions may make use of any attribute of the colloquy. Each condition has a obligation attached that will send a specified event to a particular stage.

4.0 Examples

An example plan which is used within a software development team to handle and process bug report is shown in Figure 2. It represents a Project Manager point of view on the way that a software development bug report is handled. This plan was invoked as a request from the submitter to the project team (fulfilled here by the project manager) to handle a particular bug report. The first step in the process is for the submitter to input the details about the bug. This information is kept in the colloquy's shared data area.

Once submitted, the Test Lead becomes responsible to "reproduce" the bug based on the information in the report. Failing to reproduce the bug typically means that the report is missing some detail and is therefore sent back to the submitter to supply the missing information and to submit it again. Notice that the use of loops like this make the plan non-linear and a better fit to the real world. The capability to construct such quality loops answers many of the problems discovered with linear oriented work flow systems.[21]

The request to reproduce the problem comes from the Project Manager to the Test Lead. Notice that this happens without requiring the Project Manager to be aware that a bug has been submitted. This is because the submitter has followed the plan set up by the Project Manager. This is how the model assures that proper authority is presented when the proper process is used

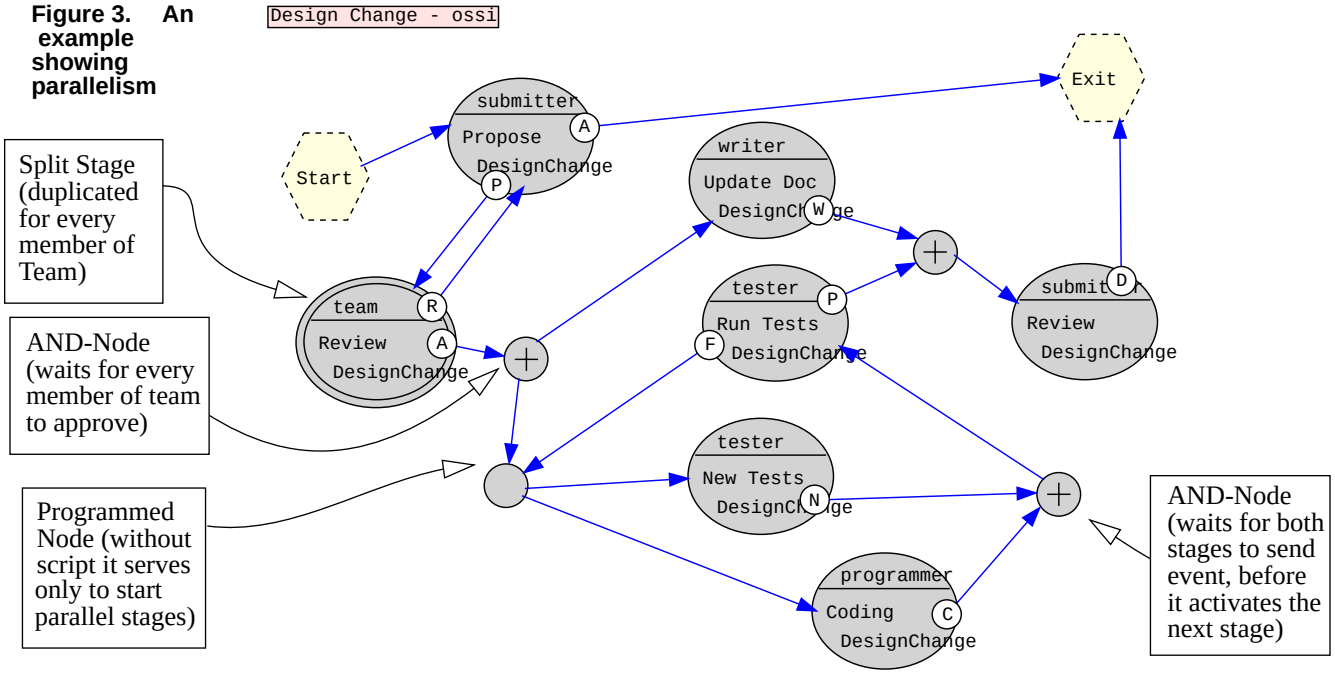

Figure 3. An example showing parallelism

The “Design Change” plan shown in Figure 3 is a good example of the proper use of parallelism. When the plan is started, the design team is asked to implement a feature. Once submitted, the review team is responsible to review the request. Since this is a split stage and because the approve option points into an AND-node, every member of the review team will need to approve it. But if any member of the team rejects the request, it goes back to the report stage. After approval by the review team, three stages are started in parallel: the doc team is asked to modify the documentation, the test team to create some tests, and the programmer is asked to implement the code changes. The three stages that would be active at this point are highlighted in the figure.

Eventually and only after the programmer and the test team have declared themselves finished with their activities will the test team be requested to run and test the new code. This stage could be completely automated with a script, or might be manual.

Once the program tests are successfully passed and the documentation changes are done, the designer is asked to review the work. The parallelism allows unrelated activities to proceed independently of each other.

4.1 No Subplan - Manual Operation

When the “Can Problem be Reproduced?” stage is activated, and there is no subplan, then the user must respond to it manually. This means that the colloquy appears on the Test Lead’s “active” list, indicating that the Test Lead has responsibility for this stage within the colloquy.

Choosing a colloquy from the active list will cause the problem report form to appear on the Test Lead’s screen. The Test Lead must respond to the request from the Project Manager by selecting either a menu item that says “Can not reproduce,” or “Reproduced,” These menu items were generated automatically from the VPL description of the plan. “Accept” & “Decline” micro-options are always automatically available in any stage so that the Test Lead can communicate whether the task will be done or not. Note that declining at this stage will bring the Project Manager into the colloquy for the first time, to resolve the problem. Eventually the Test Lead determines whether the problem is reproducible or not, selects the appropriate menu option, and the plan takes its course.

4.2 Subplans - Automating Operation

Responding to a request may be a complex process, subplanning may be used to complete the task. A subplan is no different from a plan and is described in the same VPL language. The only difference is that the possible options that the parent stage, the stage being subplanned, expects appear as exit nodes on the right in the subplan. The user can then build the subplan by adding in stages, and assigning them to people. The subplan need not be complete before it is started, because it can be edited and modified at any time. Ultimately, connecting an option from a stage to one of the exit nodes will have the exact same effect as choosing the item from the menu as it would appear in manual mode.

4.3 Automatic Subplans from Plan Templates

The Test Lead may often receive the request: “Can Problem Be Reproduced?” Manually designing and creating the plan every time could become quite tedious. Instead the Test Lead may design a plan, or save one of used plans, as a template to provide a starting configuration. This plan may then be freely modified to fit the needs of the instance.

The plan template can be set to be invoked automatically when the request is offered, or to wait until the assignee has accepted the task before creating the subplan. The latter option is so that the person might have a chance to review the details before accepting what might be a very complex task. If automatic invocation is chosen, then the assignment of subtasks could be passed on to others without the Test Lead being involved at all.

Two more options are provided in the current implementation to handle exceptions and to simplify VPL programming: The first is ability to decline the task after the plan is started. This allows one to design a plan template which examines a few parameters and continues or declines programmatically. The second is the ability to restart the plan from the start-node programmatically. this should simplify the logic needed to handle complex conditional, or when someone discovers late in the process that one of the initial assumptions was wrong. Restarting a process allows the conditional branches to be rerun from the beginning.

4.4 Key to Success

Allowing incremental automation is a key to acceptability of the model and the Regatta system. Without automation the system works like a better e-mail: a request is sent from one person to another and retains the benefit of the shared space and the history mechanism. As users become more sophisticated, or tasks more repetitive, they may automate their own tasks, by creating plans and plan templates using VPL. Tasks that are rare and are not worth the trouble to automate do not need to be automated.

It is also critical that the person who does the automating is the same person who sees the benefit from this. You automate your own tasks. the more time you put into it, the more benefit you receive. Lack of observation of this principle is cited as a reason for failure of many groupware systems.[7][8][9] A similar plea for empowering user is made in [1]. VPL empowers users to have control over their individual parts of the process, allowing them to experiment with, modify, and evaluate the processes, leading to real business process reengineering.

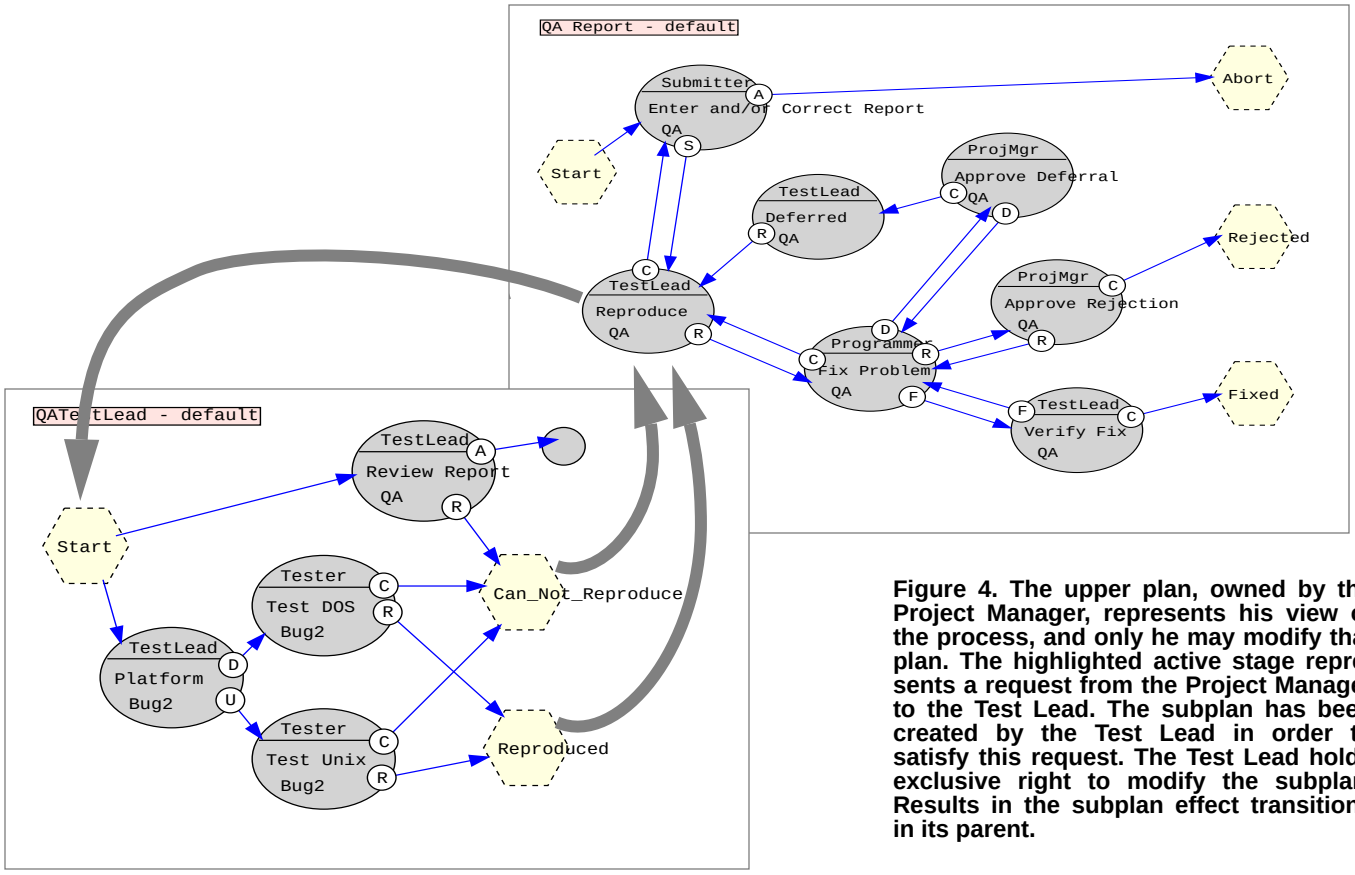

Figure 4. The upper plan, owned by the Project Manager, represents his view of the process, and only he may modify that plan. The highlighted active stage represents a request from the Project Manager to the Test Lead. The subplan has been created by the Test Lead in order to satisfy this request. The Test Lead holds exclusive right to modify the subplan. Results in the subplan effect transitions in its parent.

5.0 Other Representations

Pert Charts look superficially like VPL. The biggest difference is that the lines connecting the tasks in a Pert chart really represent dependency relationships or preconditions. A task can be started when all of the tasks that are connected before it are completed. This AND-like behavior - contrary to the VPL OR-like behavior - makes it difficult to create loops or cycles within the flow. Processes described with a Pert chart have a deterministic, all-or-nothing quality about them that makes them very unsuitable for all but the most highly structured tasks.

Work Breakdown Structure is another formalism that helps one understand the relationships of different tasks within a project. It is a hierarchical decomposition of the project into logical categories or groups. It is primarily useful for categorizing tasks. Because it lacks a way to represent parallel processes, it is considerably weaker than even pert charts.

State Diagrams are useful for linear flow processes[15]. But state diagrams would be a poor choice for modeling organizational processes due to the difficulty of representing parallelism.

State Charts[13] have been used with some success in modeling group processes.[20] While clearly an improvement over state diagrams, they do not model speech acts or communications within the group, and do not allow different parts of the plan to be owned by different people. “Hyperlines,” when used, make state charts difficult to read because connections are not made graphically explicit. While it models tasks, the person doing the task is shown in a separate representation outside of the behavior state chart.

Entity / Relationship diagrams help in understanding the relationship of organizations, people, and artifacts, but they do not provide the sequencing that is necessary for coordinating the tasks of an organization.

Data Flow diagrams give insufficient clues as to the sequencing of the events. It is difficult with data flow to describe different tasks being performed on the same artifact in parallel. Finally, one of the key benefits that information technology gives us is fast concurrent access to information regardless of location; modeling a system as moving data from one point to another works contrary to this benefit.

The Regatta approach is to allow data to be ubiquitously available to all members of a colloquy. The completion of a task is not accompanied by a loss of access to the data manipulated by that task. Thus the flow of data is far less important, allowing Regatta VPL to concentrate on coordination of behavior.

Petri Net formalism can be seen as the basis and roots of VPL. Stages are activated or deactivated by events which are quite similar to tokens. VPL can be modelled using a Petri Net. Petri Net formalism is far more general, and less compact than the VPL diagrams when modeling collaborative processes. In the Petri Net, all of the internal status would need to be modelled explicitly, making the resulting diagram impossible for an average computer user to read.

Coloured Petri Nets[16] go a long way toward simplifying the Petri Net representation of the VPL elements. Particularly, the VPL events would become simply colored tokens, and the number of duplicated constructs would be reduced. Coloured Petri Nets are not specialized for collaboration and speech acts, making them less appropriate for inexperienced users to read and understand. Regatta VPL hides a lot of the details. In many ways, the Regatta VPL could be considered an extension and specialization of coloured Petri Nets.

Information Control Nets[4] work in a way quite similar to Regatta VPL. The main difference is that VPL is simpler because it does not attempt to model the flow of documents from one task to another, or in and out of archives. Because of location independence of information this modelling of the location of documents was considered an unwarranted complication of the description of the process.

Role Interaction Nets look to be a promising way to represent collaborative behavior, but appear to be less procedural and possibly more abstract than VPL diagrams for representing common activities.

5.1 Negotiation Model

While supporting negotiation, one of Regatta Technology's strong design concepts is that it does not enforce a specific model for negotiation. This is significantly different from other systems, such as Action Workflow, where a specific negotiation model is included.

It may be true that linguists can accurately model all negotiations between two parties by a set of phases and transitions. It is also true that most parties involved in a negotiation are completely unaware of specific speech acts and phase transitions that are taking place. Speech act theory[27][28], and other linguistic theories, are interesting and important because they attempt to explain things that are not intuitive. One must question whether a theory that *explains* natural behavior is a good basis for a tool that is used in place of that behavior.

In a more natural setting, people are primarily aware of statements being made and whether the "ball is in their court". Regatta supports a face to face style communication in natural language that allows for complex levels and structures to any interaction and yet, liberates the parties involved from learning a new philosophy in order to make their moves explicit. With each statement, the "ball" may optionally be placed in the other person's court. The emphasis has been to make Regatta intuitive and easy to use by the untrained individual.

5.2 Comparison to Action Workflow

Action Workflow[22] is clearly an important system to compare to due to their strong position in the work flow industry, and also due to the fact that they offer a way of diagramming the process. Like most workflow systems, Action Workflow is not designed to be programmed by the user, does not support changes on the fly, does not support individual versions of plan templates, and has no support for incremental improvement of processes. It is not a collaborative planning tool, yet it does allow for modeling of processes where communications between individuals is represented as the key to coordination.

Because of the central place given to decision making within a work process, Regatta technology makes it exceptionally easy to create process templates with choices that lead down different paths using only the graphical editor. In many other systems, including Action Workflow, such decisions and branches are made in a much less direct way, usually involving prompting for the decision, saving a value, and then testing that value in a condition node. While not being extremely complicated, it does nevertheless require a greater sophistication as a programmer than simply drawing a new arrow coming out of a stage. When the programming is done non-graphically, it can not be seen in the graphical representation.

6.0 Conclusion and Project Status

A system that implements this model is in limited availability. The server and client both run on a Unix workstation (SPARCstation for now) with an XWindows user interface. A MS Windows supported client is expected in early 1994.

Experiments with the existing system have shown that the visual formalism is sufficiently powerful to model a wide variety of business processes. Inexperienced users are able to read the diagrams after only a few minutes of explanation about how they work. Users will typically keep a small repertoire of very generic processes, such as "Question/Answer," or "Request/Done," that are used as the basic building blocks of everyday activities. The more complicated plans are reserved for formal processes.

6.1 Results from a Typical Site

The first major deployment of the technology was to a software development team with 15 engineers and 2 managers. Their task was to merge operating system changes

from 3 different sources into a single unified version of the OS. Being such a large and complex task, this was a good candidate for use since process support more useful as the number of separate activities gets very large.

Porting began full scale in August, taking source from four different source bases. Having decided in May to use Regatta the team began to actively, use Regatta from the start of September.

The site has been using Regatta process support for 6 months. They track typically around 300 parallel colloquies through 5 distinct phases of development. They has developed a reporting tool that scans the Sybase database at a regular interval (every 15 minutes) and generates reports of the current status of all colloquies. This report is fed into XMosiac, a hypertext tool, in which live links are set up leading to Regatta colloquies. When the links are followed, the Regatta viewer is informed, and a viewer context is launched to display the desired colloquy.

It was noted that Regatta was particularly useful during the extremely active period just before an intermediate release. During this time people were interacting fairly actively through regatta. One engineer noted once being quite pleased that after making an entry reporting a problem, by the time he got to the responsible engineer's desk the problem had already been fixed. The ability for synchronous work is a key factor during times of heavy activity.

The result is a system which gets regular use and the team depends upon for maintaining their status. Keeping the status of 300 separate processes completely up to date is a job that might take a person full time. Yet with Regatta, any member of the team can have instantaneously up to date status at any time. Regatta made command assignment easier. It made tracking/dropping/reassigning of commands trivial as well as doable by everyone. It made the project status available in a consistent and regular manner. This was especially useful for seeing what remained to be finished and merged for a release.

The team made this statement: "Regatta helped formalize the process, thus improving quality. Quality was improved because no units were overlooked and QA procedures were not neglected. Regatta made it very difficult to accidently ship work that had not properly passed the QA stage."

6.2 The Regatta Vision

In summary, the model and visual language are appropriate to accomplishing the three goals:

1. Coordination: business processes may be automated; users can find tasks that are waiting for them to do; exceptions can be handled in an efficient manner.
2. Understanding Processes: discovery of the process is supported by allowing activation of incomplete plans that may be later modified on the fly; a history mechanism records what actually happened; the plan in its current state is depicted in a graphical form, with the roles, tasks, and responsibility made clear; built in help system explains new processes or changes in a process to users.
3. Change: VPL enables a greater number of users to program processes; processes are partitioned into plans at different levels to allows each user to plan their part of the process from their viewpoint, and to allows different people to control different parts of the process.

Goals two and three are complementary; while an understanding of the process helps point out areas of potential improvement, it is imperative that after a modification the system help explain the change, so that people can work effectively within the new process.

Our vision is that such a continual cycle of experimenting with new processes and observing the results will lead to what has been called a "Learning Organization"[29]. This is a new breed of organization transformed by information technology to be truly dynamic, highly efficient, and able to respond quickly to today's increasingly unpredictable external pressures.

6.3 Acknowledgments

The authors would like to thank Ron Nelson and the Equity team, Simon Kaplan and the ConversationBuilder team for discussions and early collaboration about the model; H Yoshida, T Kondo, and R Yamamoto at Fujitsu Laboratories; T Watanabe, Iwakata, S Hamano, Yamamoto, Araki, and K Fukao in Fujitsu Ltd.; and many others who helped clarifying the expressions contained herein.

7.0 Appendix / Definition of Terms

The following terms are defined to facilitate the discussion of the model features. The elements in bold are represented visually, while the other items must be entered through some form of dialog box.

❒ `system -> ([colloquy]+ [forms]+ [plan template]+)`

A **colloquy** is a context for a process. The colloquy is a shared work space which serves to hold the information needed to execute a process. A colloquy can be visualized as having a "white board" upon which the various parts of the process can read and write. The information on the white board may specify documents being worked on, attachments for specific purposes within the process, and specific field values needed for the process.

❒ `colloquy -> (id name description main-plan [subplan]+ [attribute]+ [subcolloquy-ref]+ [history-item]+ participant-list responsible-list)`

A **plan** is a formal definition of a process required to perform a specific task with specific results. A plan represents a single level, or a single viewpoint, while the entire process, or colloquy, will be composed of a collection of plans. A plan may be modified for a particular instance of a process without affecting other existing plans, or future plans.

- `plan -> (id, name description [node [arc]+]+ start-node [exit-node]+ [roles]+)`

Invoking the plan template creates a plan in its initial starting configuration. When a plan template is modified it affect only future invocations, and has no effect on plans within active processes.

A **main plan** is a plan that was used to create a colloquy.

A **subplan** is a plan that was invoked in response to a stage in another plan. A subplan is invoked within the same colloquy as its parent.

A **child colloquy** is a separate process with a different "white-board" that was started within the context of a colloquy. The parent colloquy may be waiting for the result from a child colloquy, but in all other respects they are separate processes.

The **node** is the basic building block of a plan. It represents something for a group or individual to do; it is either a task or a question. A stage will have any number of arcs placed on it in order to define how the stage will respond to various events.

- `node -> AND-node | programed-node | conditional-node | stage | split-stage`
- `stage -> (name description role-ref form-ref start-up-script shut-down-script)`

A **role** is a placeholders for the participants in a plan. Roles are used so that an individual can be assigned to a number of tasks concurrently, and so that it is easy to change the people assigned to the tasks.

A **form** is a way to present to the viewer information from the colloquy that is relevant to the current task.

An **arc** is how a particular stage is configured for its place in the plan. Selection of an option typically completes that stage of a process, and starts up another stage.

- `arc-> (option arrow)`
- `arrow -> (trigger event-to-send destination-ref)`

An **arrow** is the definition of how to send an event. Arrows can be triggered two ways: upon deactivations of the stage, and by receipt of a specific event.

- `option -> (expectation description help-message client-script)`
- `expectation -> (event-name deactivate-or-wait server-script)`

An **expectation** is a specification of what the stage is to do when a particular event is received. The expectation can be set for a particular event from a particular other stage, or from any stage. The expectation can cause the stage to be activated or deactivated.

The **Viewer** is the program run on the user's workstation to connect to the system. The viewer presents the a view customized for each user.

The **Graphical Planner** is a program that allows users to create and edit plan in VPL.

The **Form Builder** is a program that allows users to edit and create forms through which colloquy data is displayed.

8.0 References

[1] Andrew Clement, Computer Support for Computer Work: A Social Perspective on the Empowering of End Users, *CSCW 90 Proceedings*, ACM Baltimore MD, 1990

[2] Andrew Clement, Computing at Work: Empowering Action by 'Low-level Users', *Communications of the ACM*, 37(1):53-63 January 1994

[3] Thomas H Davenport, James E Short, The New Industrial Engineering: Information Technology and Business Process Redesign, *Sloan Management Review*, Summer 1990.

[4] Clarence A Ellis, Gary J Nutt, Office Information Systems and Computer Science, reprinted as reading 9 in *Computer Supported Cooperative Work*, Irene Grief ed., Morgan Kaufman, San Mateo, 1988

[5] Gartner Group, Inc. Business Process Re-engineering *SPA-210-590*, August 7, 1991

[6] Saul Greenburg, *Computer Supported Cooperative Work and Groupware*, Harcourt Brace Jovanovitch, Academic Press, 1991

[7] Jonathan Grudin, Obstacles to user involvement in software product development, with implications for CSCW, reprinted in [6]

[8] Jonathan Grudin, Why CSCW Systems Fail, *Proceedings of the 1988 Conference on Computer Supported Cooperative Work*, ACM, p85-93, Portland Oregon, 1988

[9] Jonathan Grudin, Groupware and Social Dynamics: Eight Challenges for Developers, *Communications of the ACM*, 37(1):93-105 January 1994

[10] Keith Hales, Mandy Lavery, *Workflow Management Software: The Business Opportunity*, Ovum Ltd. December 1991

[11] Michael Hammer, W G Howe, V J Kruskal, I Wladawski, "A very high level programming language

for data processing applications," *Communications of the ACM*, 20(11):832-840, Nov 1977

[12] Michael Hammer, Re-engineering Work: Don't Automate, Obliterate, *Harvard Business Review*, July/ August 1990

[13] David Harel, On Visual Formalisms, *Communications of the ACM*, 31(5):514-530, May 1988

[14] Carl Hewitt, Offices are Open Systems, *ACM Transactions on Office Information Systems*, 4(3):271-287, July 1986

[15] Robert J. K. Jacob, A State Transition Diagram Language for Visual Programming, *IEEE Computer*, 18(8):51-59, August 1985

[16] Kurt Jensen, *Coloured Petri Nets*, Springer Verlag, Berlin, 1992

[17] Simon M Kaplan, William J. Tolone, Douglas Bogia, and Celsina Bignoli, "Flexible, active support for collaborative work with Conversation Builder", *Proceedings of the 1992 Conference on Computer Supported Cooperative Work*, ACM, 1992

[18] Simon M Kaplan, Alan M Carroll, Kenneth J MacGregor, Supporting Collaborative Processes with ConversationBuilder, *Proceedings ACM Conference on Organizational Computing Systems*, p69-79, November 1991

[19] Simon M Kaplan, Keith D Swenson, Operating System Suport for Collaborative Work, *Proceedings for the Second International Workshop on Object Orientation in Operating Systems*, September, 1992

[20] Marc I Kellner, Software Process Modeling Support for Management Planning and Control, *1st International Conference on the Software Process*, Redondo Beach, CA, October 1991.

[21] Thomas Kreifelts, Elke Hinrichs, and Karl-Heinz Klein, Experiences with the Domino Office Procedure System, *Proceedings of the Second European Conference on Computer Supported Cooperative Work (ECSCW '91)*, p117-130, Amsterdam, September 1991

[22] Raul Medina-Mora, Terry Winograd, Rodrigo Flores, Fernando Flores, The Action Workflow Approach to Workflow management Technology, *Proceeding of the 1992 Conference on Computer Supported Cooperative Work*, ACM, 1992

[23] Michael S Scott Morton, *The Corporation of the 1990s, Information Technology and Organizational Transformation*, Oxford University Press, New York, 1991

[24] John K. Ousterhout, Tcl: an Embeddable Command Language, Computer Science Department, UC Berkeley. Information on this can be retrieved from the Sprite Project, at sprite.berkeley.edu.

[25] Sunil K Sarin, Kenneth R Abbott, Dennis R McCarthy, A Process Model for Supporting Collaborative Work, *Proceedings ACM Conference on Organizational Computing Systems*, November 1991

[26] Michael Schrage, *Shared Minds: The New Technologies of Collaboration*, Random House, New York, 1990

[27] John R Searle, *Expressions and Meaning: Studies in the Theory of Speech Acts*, Cambridge University, Cambridge, 1979

[28] John R Searle, A Classification of Illocutionary Acts, *Language In Society*, 5 1-23, 1976

[29] Peter M. Senge, *The Fifth Discipline: The Art and Practice of the Learning Organization* Doubleday/ Currency, New York, 1990

[30] Nan C. Shu, FORMAL: A Forms-Oriented Visual Directed Application Development System, *IEEE Computer*, 18 (8): 38-49, August 1985

[31] Paul Strassman, *Information Payoff: The transformation of work in the electronic age.* Free Press, New York, 1985

[32] Keith D Swenson, The Regatta Project, *Proceedings of the First International Conference in Technologies and Theories for Human Cooperation, Collaboration*, and Coordination, Applica '93, March 1993

[33] Keith D Swenson, A Visual Language to Dscribe Collaborative Work, *Proceeding of the International Workshop on Visual Languages*, Bergen Norway, August 1993

[34] Keith D Swenson, Visual Support for Reengineering Work Processes, to appear in *Proceedings of the Conference on Organizational Computing Systems*, ACM press, Milpitas California, Nov 1993.

[35] Workflow, Groupware, and Re-engineering:, *IT Horizons*, 1(3):1-11, September 7, 1992

[36] Michael Zisman, *Representation, Specification, and Automation of Office Procedures*, PhD Thesis, University of Pennsylvania, 1977

[37] Michael Zisman, "Office Automation: Evolution or Revolution", Sloan Management Review 19(3):1-16, Spring 1978

[38] Shoshana Zuboff, *In the Age of the Smart Machine*, Basic Books, New York, 1988.