

A Visual Language to Describe Collaborative Work

Keith D. Swenson

Fujitsu Open Systems Solutions, Inc.

In order to satisfy goals for developing collaboration software, a powerful yet simple visual language has been developed for use by end users in describing plans for work activities. The approach presented is unique because the model allows for collaboration during the planning process; different people are responsible for different parts of the plan. Process plans may be modified on the fly, to allow handling of exceptions and other changes. Policies may be created that automatically create a plan for a user in specific situations. Plans at different levels represent the viewpoint of the people responsible for those plans. These visual representations of plans are then used directly to facilitate the coordination of those activities.

1. Introduction and background

Much of the research into visual languages has concentrated upon finding representation for programs that will be executed by machines. This paper covers a decidedly different area by attempting to find a visual representation for the processes that human organizations go through.

Coordination of human tasks is less concerned with the actual tasks that are being done, and more concerned with the communications: how we know when to start, when others are finished, and what we are supposed to be doing. Organizations coordinate work by designating dedicated people as leaders who coordinate activities primarily by calling meetings. So, in a certain sense, groupware is aimed at reducing the number of leaders and the frequency of meetings. This by itself should warrant serious consideration.

More recently, attention has been drawn to the concept of Business Process Reengineering (BPR)[2][6][7]. This is the process of improving the efficiency of an organization by redesigning tasks to make better use of information technology.[17][25] While some cases have met with tremendous results, BPR has not been generally adopted because it is exceedingly difficult to discover and redesign

organizational processes. What is needed is a tool to support this activity.

1.1 Earlier approaches

Work flow software attempts to coordinate by routing documents from one person to the next. Studies have shown that this approach is woefully inadequate for real world situations.[16] With more than a few people, the exceptions to the flow outnumber the normal cases.[21] The routing list must be continually changed because of personnel and job positions changes, vacation or sick leave, and other specialized situations.

A better model of work process is provided by systems that can accurately reflect the large amount of the exceptional conditions that must be considered. Usually a specialist is required to discover and program the model. The results are monstrously complex flow plans often involving a myriad of details that must be entered correctly before the flow will work. The organization must make a tremendous investment beforehand to develop the process, and is saddled with maintaining the complex work flow in order to keep up with all the minute changes in the organization. The benefit from this approach is rarely worth the hefty initial investment except in cases where the process is very formal or mistakes are very expensive. The real world is just much more rich and complex than we realize.

Some workflow packages have taken the approach that the workflow should be guided by a set of rules. The first problem is that the rules in existing processes are embodied in the implicit knowledge of the workers, who are not themselves explicitly aware of them.[26] Observation of worker actions will not reveal the rules they use. The second problem is that large rule-bases become unmanageable. Finally, it has been pointed out that in the office environment there are a lot of contradictory rules.[9] Which rules a team uses depends in a non-trivial way upon what has led up to the current situation.

1.2 Reasons for the difficulties

Much of the difficulty in modeling organizational work is because we learn by trial and error, work knowledge has never been expressed in an external form, and that knowledge is for the most part inexpressible.[26]

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

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

All of the approaches above fail because they attempt to model a system from an external viewpoint; they do not include the process definition activity in the model itself. This means that while enacting the processes, the definition of the process is treated somehow as if it is omniscient. The person who designed the process does not appear, and the authority of the commands are absolute. A person commanded by the system to perform a particular action has no way to question this command, nor any controlled way to modify the process.

Though some systems have the capability to allow modification of the process on the fly, the process is most often defined in a monolithic block such that anyone with the ability to change any of the process has the ability to change all of the process. This makes it very difficult to control change. Assuring that some parts of the plan do not change is just as important as allowing other parts to be changed.

2. Proposed approach

The Regatta approach models requests for work rather than the work itself. Groups are composed of dyadic relationships. A request is always from one person (or group) to another. Each request has one or more options for the recipient of the request. A declarative, being a speech act that in its utterance effects the state of the group, is an observable external behavior that all parties can agree upon.[24][19]

The visual language description of a level of the process is called a *plan*. The plans that implement a process, and the data that they share is called a *colloquy*. The template for a plan is called a *policy*. Policies and plans are created and edited using a graphical picture-oriented notation called VPL (Visual Process Language). In VPL, a plan is composed of *stages* which represent individual requests. Stages have *options* which can activate other stages.

The options on a stage represent the expected declaratives to conclude the task. A person can plan a request to be made automatically after other requests have been satisfied. Requests to yourself become reminders of things to be done. Plans can be made and changed on the fly as needed.

The recipient of the request can also make a plan with the goal of satisfying the request. The recipient owns and can modify this sub-plan, which is composed of requests to himself or other people. If the request is commonly received, the recipient can set up a policy to be automatically invoked to implement the plan. The entire process is composed of many plans at different levels that work together in a way that is similar to the organization.

2.1 Project goals

The Regatta group was formed in 1991 to develop software to support workgroups and to aid in reengineering work processes[22][23]. To achieve this, the Regatta project has set three specific goals:

- ❖ To make software that supports the *coordination* of activities of different members of a group by automating work processes.

- ❖ To make software that allows every individual in a group to gain a better *understanding* of how their group or organization works.
- ❖ 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.

2.2 Requirements for the model

In order to meet these goals, we need a system and a visual programming language that meet the following requirements:

- ❖ An average computer user is able to draw pictures of their processes.
- ❖ The same visual representation is used to show process status.
- ❖ The process can be modified dynamically.
- ❖ The process definition need not be complete before it is enacted.
- ❖ The system should allow different people to have different processes for the same request.
- ❖ The model must support multiple levels of abstraction.
- ❖ People doing the work should be in control of the process.
- ❖ The system should accurately communicate the authority of a request.
- ❖ The process of discussion and negotiation about the request should be supported.
- ❖ The model should support in a straightforward manner delegation of responsibility.
- ❖ The model must be able to automate processes.
- ❖ The system must support the use of external tools.
- ❖ The model must record what actually happened, not just what was supposed to happen, so that the history can be used to improve the process.

3. Plans and subplans

An example plan to handle and process bug reports is shown in Figure 1. It represents a project manager point of view. The first request is from the project manager to the submitter to input the details about the bug. The submitter is given two options: submit or abort. After submission, the test

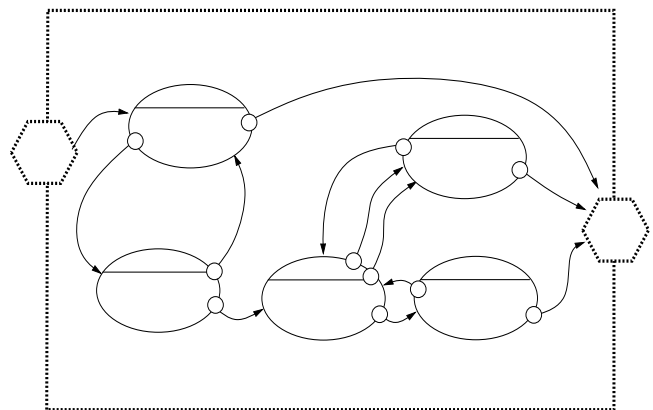


Figure 1. An example plan: quality assurance

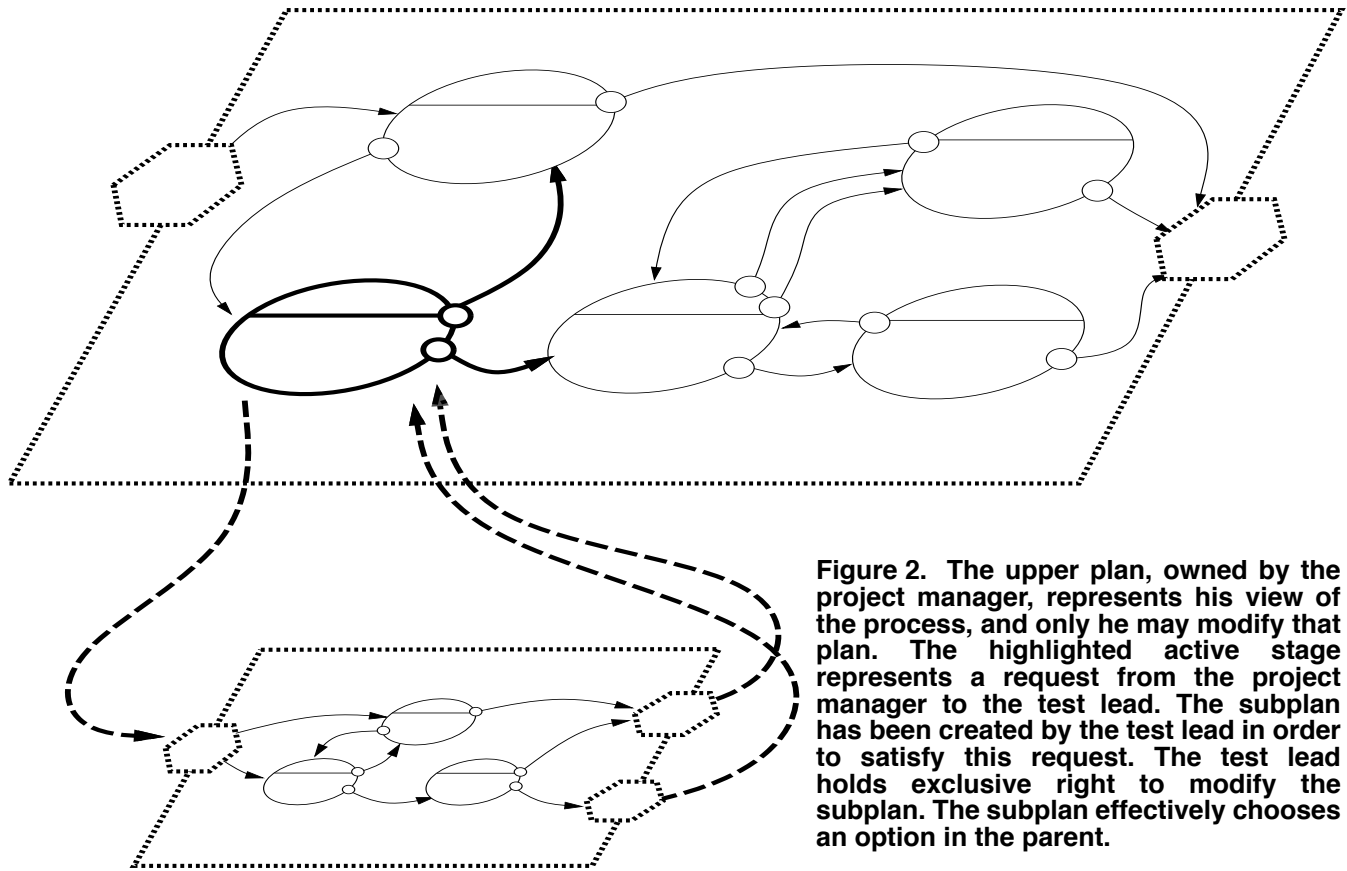


Figure 2. The upper plan, owned by the project manager, represents his view of the process, and only he may modify that plan. The highlighted active stage represents a request from the project manager to the test lead. The subplan has been created by the test lead in order to satisfy this request. The test lead holds exclusive right to modify the subplan. The subplan effectively chooses an option in the parent.

lead is responsible to “reproduce” the bug, that is, to verify that the anomalous behavior can be repeated from the instructions in the report. Failing to reproduce the bug usually means that the report is missing some detail and is therefore sent back to the submitter to supply the missing information. The submitter then resubmits the report. Notice that the use of loops like this makes the plans non-deterministic, which is a better fit to the real world and answers many problems discovered with linear work flow systems.[16]

The request to reproduce the problem comes from the project manager to the test lead, not from the submitter. Proper authority is preserved because the submitter has followed the project manager’s desired process.

Manual operation: When the “Can Problem be Reproduced?” stage is activated, and there is no subplan, then the test lead must handle it manually. The options “Can Not Reproduce,” and “Reproduced” appear as menu items generated directly from the VPL. “Accept” & “Decline” are options automatically available to support communication about the request. Manual selection is appropriate for small teams or infrequent tasks.

Automation through subplans: If the test lead wishes to fulfill the request by making requests to others, a subplan can be immediately drawn up to do this starting as shown in Figure 3. The options of the request appear as exit nodes on the right. The test lead can then build the plan by adding in stages, and assigning them to people. The plan need not be

complete before it is started, because it can be edited and modified at any time. Ultimately, sending an event to one of the exit nodes will have the exact same effect as manually choosing the menu item for that option (see Figure 2).

Automatic subplans from policies: If the request is made frequently, the test lead may save one of his plans as a policy, and have it automatically invoked whenever he receives such a request. The policy is just a template for a plan, which can then be freely modified to fit the needs of the instance. The plan for the entire process emerges as people define policies for their own parts of the process.

Key to success: Allowing incremental automation is a key to acceptability of the model and the system. Without automation the system works like e-mail: a request is sent from one person to another while retaining the benefit of the

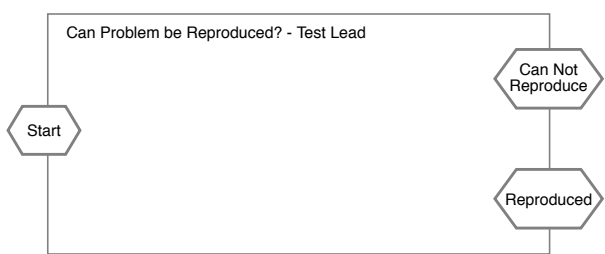


Figure 3. An empty subplan diagram

shared space and the history mechanism. Then, as the process becomes more apparent, or tasks more repetitive, users can automate their own tasks by creating plans and finally policies using VPL. Tasks that are rare and are not worth the trouble to automate can be handled manually.

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. This is critical.[4][5][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.

4. Visual elements

Users use a program, Graphical Planner, to create and edit policies and plans. *Stages* (steps in the process) are represented in VPL by an ellipse containing the description of the request and the role to receive the request. The *role* is a placeholder which is filled when activated with a list of user or group names. A split stage is another kind of stage that automatically duplicates itself in parallel for every person assigned to the role. The stage also has associated with it a *form*, which is used to display, edit, and control data associated with the colloquy.

On the edge of the ellipse are *options* which appear as a circle with an arrow and are more completely labeled below the stage. When the stage is active, the options appear as menu items. Normally when the user chooses an option, the current stage is deactivated, and the stage that the option points to is activated. Alternately the option might leave the stage it is on active which would be denoted in VPL by reversing the colors of the circle; in this case both stages would be active at the same time.

The receiving stage is activated if the *activate* event is sent. Other events may be sent by labeling the arrow with the event name to send, which would trigger any similarly named options on the receiving stage. Alternately the option might not send an event which would be denoted by the lack of an arrow.

The owner of a plan creates the stages and options and is the only person allowed to modify them. Yet for synchronization purposes other plans might wish to be informed about changes in a plan in such as way to be able to automatically respond to such changes. For this reason, any participant of a colloquy is allowed to place obligations on stages in a plan. An *obligation* will send a specified event when the stage it is placed on activates or deactivates. Obligations can be used in an ad-hoc manner to set up dependencies between tasks that are not in the same plan.

While stages represent steps in the process which wait for users to choose an option, there are other elements which do not appear to the users because they are handled automatically. Start nodes (a hexagon on the left) activate stages when a plan is created. Exit nodes (hexagons on the right) send a specified event to the parent plan. Programmed nodes (a small circle) are used to simply execute a script when activated. Condition nodes test a set of conditions and send

corresponding events to other nodes or stages. A timer node waits a specified amount of time before sending its event onward.

When two or more options are pointing to the same stage, an activate event from any one of them will activate the stage. This OR-like behavior is common for all VPL elements except for the AND-node (represented as a small circle with a plus in it) which wait to receive events from all of the stages before it send.

A scripting language is provided by the system to allow automatic manipulation of colloquy data, or to control external tools.[18] Each stage has a user defined script that is executed whenever the stage is activated, and another for when it deactivates. Each option has a script that is executed if the user chooses that option. Programmed nodes and condition nodes use the same scripting language.

A more complete description of the visual elements can be found in [23].

5. Examples

The “Feature Design” policy shown in Figure 4 is a good example of the proper use of parallelism. When the plan is started, the designer is asked to specify the feature to be implemented. After the designer submits the review team is requested to review the design. Since this is a split stage and the approve option points into an AND-node, every member of the team will need to approve individually. But if any member of the team rejects it causes a return to the design stage. After approval, three stages are started in parallel: the doc team is asked to modify the documentation, the test team is asked 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.

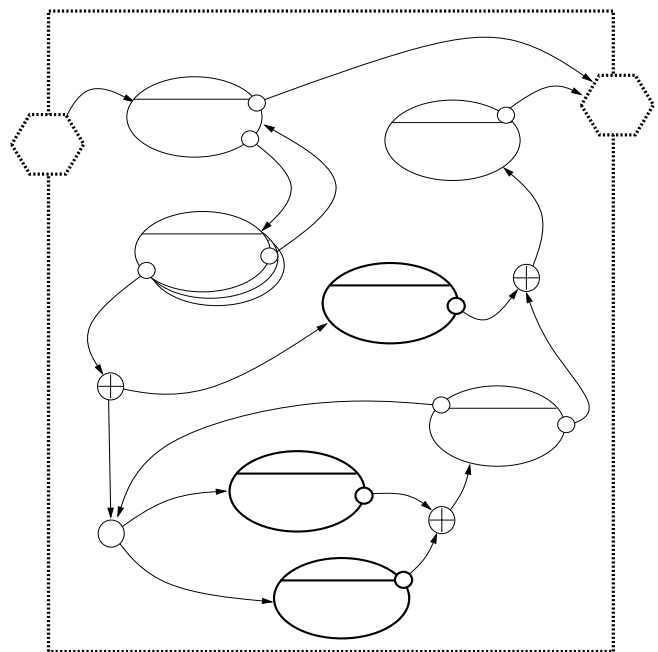


Figure 4. An example showing parallelism

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

Once the program passes the tests 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.

The “Purchase Order” policy (Figure 5) shows an example of how a company might process a purchase order. At the company level, the department is requested to approve the purchase and it may do so in any manner it sees fit. In this example the department director only gets involved on larger purchases, otherwise delegating this responsibility to the team manager. Different departments may implement very different policies for approval. Combining the stages from the two plans into a single flat diagram would simplify the picture, but would deprive the department of the control over its own approval process.

6. Comparison to 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.

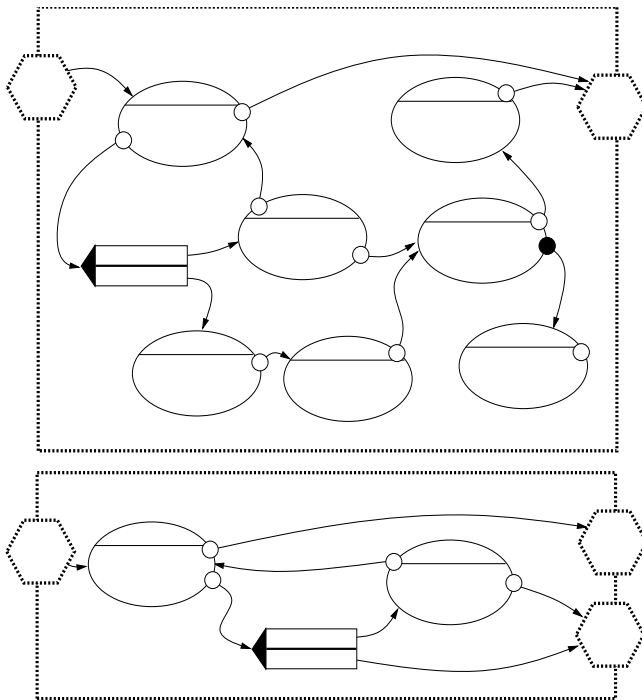


Figure 5. An example using a sub-plan

Work Breakdown Structure is another formalism that helps one understand the relationships of different tasks within a project. It is primarily 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[10]. But state diagrams would be a poor choice for modeling organizational processes due to the difficulty of representing parallelism.

State Charts[8] have been used with some success in modeling group processes.[15] While clearly an improvement over state diagrams, they do not model dyadic relationships, and do not allow different parts of the plan to be owned by different people. “Hyper-lines,” 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 work 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[11] 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.

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

7. Results

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. When building new policies, users were generally able to create single level policies without trouble. Users typically implemented “flat” policies that included all of the details of the process at a single level, and only after some practice would split the process up into levels.

Early experience has shown that 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 policies are reserved for formal processes.

A system that implements this model is in beta test. 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.

7.1 The Regatta Vision

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”[20]. 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.

7.2 Acknowledgments

The author would like to thank Robin Maxwell, Toshikazu Matsumoto, and Bahram Saghari for help in developing and refining these ideas, and for developing the system which implements this model; and Simon Kaplan and the ConversationBuilder team[12][13][14] for discussions about the usability of the model, and for implementing an early prototype in ConversationBuilder.

8. 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] Thomas H Davenport, James E Short, The New Industrial Engineering: Information Technology and Business Process Redesign, *Sloan Management Review*, Summer 1990.
- [3] Saul Greenburg, *Computer Supported Cooperative Work and Groupware*, Harcourt Brace Jovanovitch, Academic Press, 1991
- [4] Jonathan Grudin, Obstacles to user involvement in software product development, with implications for CSCW, reprinted in [3]
- [5] Jonathan Grudin, Why CSCW Systems Fail, *Proceedings of the 1988 Conference on Computer Supported Cooperative Work*, ACM, p85-93, Portland Oregon, 1988
- [6] Keith Hales, Mandy Lavery, *Workflow Management Software: The Business Opportunity*, Ovum Ltd. December 1991
- [7] Michael Hammer, Re-engineering Work: Don't Automate, Obliterate, *Harvard Business Review*, July/August 1990
- [8] David Harel, On Visual Formalisms, *Communications of the ACM*, 31(5):514-530, May 1988
- [9] Carl Hewitt, Offices are Open Systems, *ACM Transactions on Office Information Systems*, 4(3):271-287, July 1986
- [10] Robert J. K. Jacob, A State Transition Diagram Language for Visual Programming, *IEEE Computer*, 18(8):51-59, August 1985
- [11] Kurt Jensen, *Coloured Petri Nets*, Springer Verlag, Berlin, 1992
- [12] Simon M Kaplan, William J. Tolone, Douglas Bogia, and Cel-sina Bignoli, “Flexible, active support for collaborative work with Conversation Builder”, *Proceedings of the 1992 Conference on Computer Supported Cooperative Work*, ACM, 1992
- [13] Simon M Kaplan, Alan M Carroll, Kenneth J MacGregor, Supporting Collaborative Processes with Conversation-Builders, *Proceedings ACM Conference on Organizational Computing Systems*, p69-79, November 1991
- [14] Simon M Kaplan, Keith D Swenson, Operating System Support for Collaborative Work, *Proceedings for the Second International Workshop on Object Orientation in Operating Systems*, September, 1992
- [15] Marc I Kellner, Software Process Modeling Support for Management Planning and Control, *1st International Conference on the Software Process*, Redondo Beach, CA, October 1991.
- [16] 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
- [17] Michael S Scott Morton, *The Corporation of the 1990s, Information Technology and Organizational Transformation*, Oxford University Press, New York, 1991
- [18] 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.
- [19] John R Searle, *Expressions and Meaning: Studies in the Theory of Speech Acts*, Cambridge University, Cambridge, 1979
- [20] Peter M. Senge, *The Fifth Discipline: The Art and Practice of the Learning Organization* Doubleday/Currency, New York, 1990
- [21] Lucy Suchman, The role of common sense in UI design, *Highlights on the International Conference on Office Work & New Technology*, Cleveland, 1983.
- [22] Keith D Swenson, The Regatta Project, *Proceedings of the First International Conference in Technologies and Theories for Human Cooperation, Collaboration*