The Chautauqua Workflow System

Clarence (Skip) Ellis* and Carlos Maltzahn*
Department of Computer Science, University of Colorado
Boulder, Colorado, USA 80309-0430

Abstract

Chautauqua is an exploratory workflow management system designed and implemented within the Collaboration Technology Research group (CTRG) at the University of Colorado. This system represents a tightly knit merger of workflow technology and groupware technology. Chautauqua has been in test usage at the University of Colorado since 1995. This document discusses Chautauqua - its motivation, its design, and its implementation. Our emphasis here is on its novel features, and the techniques for implementing these features.

* This work was partially supported by NSF grant IRI-9307619

1. Introduction

Workflow management systems have been defined as “technology based systems that define, manage, and execute workflow processes through the execution of software whose order of execution is driven by a computer representation of the workflow process logic” [23]. They have been called collaboration aware groupware. Whereas most groupware has been criticized because it is not organizationally aware, workflow has been criticized because of its typically inflexible and dictatorial nature compared to the way that office workers really accomplish tasks [14]. Thus, this document introduces Chautauqua, an experimental research system which merges features of groupware and workflow to produce a flexible collaboration management system which is organizationally aware.

2. Related Work

There has been other work attempting to bring together workflow and groupware technologies. Industrial research labs and product teams have made significant steps forward. The interesting work and products of Winograd and Flores based upon speech act theory suggests that any interaction can be viewed as a “conversation” with a protocol structure that can be modeled as a workflow[13]. The coordinator was a product emerging from this work that had this protocol notion built in. Likewise, the action technologies workflow product [24] is based upon speech act theory, and describes every workflow as beginning with a request, and successfully completing with an agreement of satisfaction. The GMD research center has a long history of interest in workflow and collaboration technology. Several research workflow systems were implemented and experimented with valuable learning results[17]. Recently, GMD has introduced a very flexible shared workspace system on the world wide web internet [3]. Work at a number of universities has explored some instances of merging of workflow and groupware, including work at the Universities of Milano, Illinois, Erlangen, Toronto, and MIT.

At Xerox PARC in the 1970s, one of the authors, Ellis, was involved in the development of the Officetalk system [8], the first of the workflow systems developed using personal workstations, local area network, and graphical user interfaces. This led him to address various other aspects of office work and organizational design. The original Information Control Net (ICN) model was also developed by this author in the 1970s[7], which led to a family of related modeling studies.

3. The Chautauqua Model

The Chautauqua model is based upon the extended Information Control Network (ICN) process model. This section describes the standard features of ICNs without exception handling. The next section describes conflict management, dynamic change and exception handling, and how the standard features are extended to accommodate these novel features.

3.1 The Abstract Structure of Work: Models

In general, a model is a simplified abstraction of reality. It sometimes allows one to see the patterns of behavior of important parameters of the real system by ignoring exogenous variables of lesser concern. Models can be useful for developing an understanding of a complex system, for performing mathematical analyses, for human communication about the system, and to allow simulation studies of the system. *A workflow model* is an abstraction capturing pertinent aspects of a work environment, and work processes within an organization or workgroup. It typically is concerned with aspects of the organization modeled by tasks, actors, roles, activities, and information objects. A workflow model is most convenient if it is supported by a computerized graphical implementation. A computerized workflow system typically allows the creation, alteration, and simulation of workflow. This editor should present the model in a form which is understandable to the non-specialist, and easy to manipulate. Besides being simple to understand, a good model should be mathematically tractable. Thus it should be precisely defined, and there should be meaningful analyses that leverage this mathematical notation. These analyses should be invoked and controlled via the model editor. Our Chautauqua thesis is that if the workflow modeling system fulfills the above criteria, then it is not only useful as an organizational pre-design tool, but is useful for exception handling, dynamic change, and general information sharing during all phases of workflow modeling and enactment.

3.2 The Information Control Net Model

The Information Control Net is an organizational activity model with characteristics similar to timed, colored Petri nets [11]. It generalizes Petri nets by adding a complementary data flow model, generalizing control flow primitives, and simplifying semantics so that the model is intuitive and useful to office procedure designers. A summary of office models, in general, is presented by Sheth [21]. In this section we describe the basic ICN. Basic ICNs have been used in many contexts to describe structured work, but exhibit the usual shortcoming when they attempt to fully prescribe an office procedure that is executed in many different ways under many different circumstances. Office models must be sufficiently flexible to capture the intent of a procedure, but not so prescriptive that they are a barrier to the office worker.

Experience with the basic ICN model suggests that workflow must be cognizant of organizational goals and structures, individual skills and preferences, as well as the sustaining social environment. These are the primary modeling research issues that must, in the longer term, be addressed. Motivation for much of this research can be seen by considering exception handling, which is a prevalent activity within offices. For example, when an exception arises such that the next activity cannot be executed, people who know the goals of their information processing, can derive alternative activities to attain the same goals. They can use their knowledge of the organizational structures to obtain proper approval of these alternative activities. They can use their social network to expedite information exchange if necessary. If the workflow system is aware of the goals and preferences, then it can assist in this process of exception handling. Thus, longer term research includes the imbedding of social and organizational sub-models within the ICN model. Coherent definitions of subgoals, of conflicting goals, of group goals, and of subgroup goals are now being incorporated into ICNs [22]. We observe that people do a lot of problem solving, and that a lot of interaction with colleagues is necessary for this. Thus, we are also in the process of implementing synchronous and asynchronous distributed meeting facilities within Chautauqua.

The basic ICN is a simple, but mathematically, rigorous formalism intended to model organizational procedures [11]. It is intended to represent control flow and data flow; which activities precede which other activities; other relations indicate which data repositories are input or output for which activities. ICNs have been studied in universities [5], and used in industry [4]. They have been shown to be valuable for capturing organizational procedures, for mathematical analyses, and for simulation. Some of the documented analyses of ICNs include throughput, maximal parallelism, and reorganization, and streamlining [5].

An ICN graph is composed of one *start activity*, one *end activity*, a set of intermediate *activities*, *disjunctions*, *conjunctions*, *actors*, *roles*, and directional arcs to interconnect instances of these nodes.

Among activities, disjunctions, and conjunctions, an arc represents precedence; if activity *A* leads to activity *B* (i.e., *(A,B)* is an edge in the graph), then activity *A* must always occur before activity *B*. Suppose we have activities *A*, *B*, and *C*. Let *X* be a disjunction with arcs *(A,X)*, *(B,X)*, and *(X,C)* in the graph (*X* is a *disjunctive join*); then activity *C* can occur after either activity A or *B* have occurred. Similarly, if *X* is a disjunction and we have *(A,X)*, *(X,B)*, and *(X,C)* (*X* is a *disjunctive split*), then after *A* has occurred, either *B* or *C* can occur. Conjunctions use conjunctive flow logic as opposed to the disjunctive logic of disjunctions; if *Y* is a conjunction and the graph contains *(A,Y)*, *(B,Y)*, and *(Y,C)* (*Y* is a *conjunctive join*), then C can occur only after both A and B have occurred (cf. the traditional join concurrency control flow construct). If *Y* is a conjunction and the graph contains *(A,Y)*, *(Y,B)*, and *(Y,C)* (*Y* is a *conjunctive split*), then both *B* and *C* can occur after *A* has occurred (cf. the fork construct). The ICN formalism allows one to collapse joins and splits of the same flow logic into a single node.

An arc going from an actor to a role defines the role of that actor. An actor can have multiple roles and roles can be shared by multiple actors. An arc going from a role to an activity defines the responsibility of a role.

An ICN is *well-nested* iff all out-going paths of a split are joined by a join of the same logic as the split, where a path is a well-nested ICN. The well-nested property makes mathematical analyses of consistency and correctness much more tractable [16]. For the rest of the paper we assume ICNs to be well-nested.

ICNs have a graphical representation as well as a mathematical tuple-set representation. As an illustration, Fugure 1 shows how an ICN might represent order processing within a typical organization. The triangle-shaped nodes are start and end activities; the large, open circles are activities; the small, open circles represent disjunctions; small, filled circles stand for conjunctions. Note that multiple out-going arcs of a disjunction are labeled. These labels represent choices that are made during the previous activity. For example in figure 1, the outcome of an order evaluation can be either to reject or accept an order. Squares represent roles of employees and the stars represent actors. In figure 1, Oscar plays the role of a credit clerk who is responsible for checking references of customers and evaluate their orders.

3.3 Enacting ICNs

In an organization, work is typically generated by a request for something. We call this request a *workcase*. A workcase can be viewed as a process that enacts an ICN. Each workcase is time stamped at its creation and contains information about the requestor, a deadline and an indication of urgency. Each workcase creates one initial *token*. A token represents an activity thread within the workcase. Each token has a *location* which can be any activity, disjunction or conjunction of the enacted ICN. The location represents the state of an activity thread. The initial token of a workcase is located at the start activity. Without loss of generality, we insist that the start (and end) activity be unique.

When a token is created it is associated with a new instance of the sample *form* defined by the enacted ICN. A form represents a domain specific document with a list of *fields*. We assume activities as being sufficiently reflected by a user manipulating field entries. We distinguish between *local* and *global* fields. Entry to local fields are local to the associated token, while the content of global fields are local to the associated workcase, i.e., global to all tokens of that workcase.

A token carries a *decision list* which contains routing information for a *disjunctive split tree*. A disjunctive split tree is a fraction of an ICN that is tree shaped and contains no nodes other than disjunctive split nodes. The decision list is always created by the last activity prior to a disjunctive split tree and contains routing information for each disjunctive split node.

A token has a *status* which represents the state of a token within locations. There are five possible values for the token status: the initial status of a newly constructed token at an ICN start node is *“new”*. A token returns to the status “new” whenever its location changes to an activity. When a user looks at the form associated with a token, its status changes to *“active”*. A user can suspend work on a form in which case the token status changes to *“suspended”*. After a user declares work on a form as finished, the token status becomes *“completed”*. If the location of a token changes to a conjunctive join and the token does not match with a certain number of other tokens, the token status is set to *“blocked”*. If the token becomes “completed” at an ICN end node, it is destructed.

In the example of figure 1, two tokens are located at the activities “Reference Check” and “Inventory Check”. Both tokens belong to workcase #27 which was created for requester “David” (in this case a customer). One token has the status “new” and the other has the status “active”.

Whenever a token’s status changes from the status “active” to the status “completed”, its *history list* is updated. Each entry of the history list contains the time when the token was activated, the time when it was completed, the activity that was completed, and the actor who performed the activity.

The central notion of ICN enactment is a set of operators that are applied to one token or a set of tokens. An operator is either initiated by a user, called by another operator, or triggered by a certain condition. The result of an operator is a change to a token’s location or a token’s status (see figure 2). The *“activate”* operator is initiated by the user by looking at a token’s form. It signifies that work on this form has started. The “activate” operator is only applicable to tokens with status “new”, and it changes the token status to “active”. The operators *“suspend”*, *“resume”*, and *“complete”* with their obvious meanings are also user initiated and have analogous restrictions and effects

The *“schedule”* operator is triggered whenever a token’s status is set to *“completed”*. It repeatedly changes a token’s location to the location’s successor until the location is either an activity or a conjunctive join with not enough matching tokens. If this *final loca-*

tion is an activity the token status is set to “new”, otherwise it is set to “blocked”. Until a token arrives at a final location it might pass through various kinds of ICN nodes which sometime require the “schedule” operator to take specific actions: at a disjunctive split the operator decides among the outgoing branches based on the token’s decision list. At a conjunctive split the operator creates copies of the current token and applies the “schedule” operator on each of them. At a conjunctive join the operator tries to match the token with other token of the same workcase. If a number of input tokens match and that number is equal to the number of input arcs, the “schedule” operator applies the *“merge”* operator on all matching tokens. Assuming that the number input arcs and the number of matching tokens is n. The “merge” operator merges the forms and history lists of all n tokens, assigns the merged form and history list to the n^{th} token, destructs all n-1 tokens, and sets the status of the n^{th} token to “completed”. After the token arrives at a final location the “schedule” operator calls the *“dispatch”* operator which assigns the token to an actor based on the ICN.

4. The Novel Features of Chautauqua

So far we have introduced basic concepts of the ICN formalism and its enactment. The Chautauqua extends these concepts in various ways which are discussed below. This leads to an extended token status transition diagram which is shown in figure 3.

4.1 Conflict Detection and Management

Chautauqua provides a value conflict detection and management mechanism which is an extension of forms and the “merge” operator. We extended the form of each token such that it associates a *conflict list* for each form entry. When two or more tokens are merged and the corresponding form entries have conflicting values, the form of the resulting token lists all conflicting values in the corresponding conflict lists. The Chautauqua system provides interactive means of conflict resolution.

4.2 Dynamic Change

Our approach to dynamic change is to make structural changes to an ICN part of the ICN enactment. We introduce two operators that integrate structural changes with enactment: the *“edit”* operator performs structural changes that result in syntactically correct ICNs. The *“fall out”* operator which collects tokens that have lost their location due to structural changes, deletes their location references, and sets their token status to “blocked”. We call a token that lost its location a “fall out” token.

The design of the Chautauqua model carefully distinguishes between structural concepts and enactment concepts. This allows us to use long conventional transactions with write locking for structural changes without blocking enactment: A token can change its location even when both locations are part of a locked ICN. In the current Chautauqua system, every structural change locks the entire ICN. Thus, only one structural change per ICN is possible. Future versions of Chautauqua will allow parallel structural changes on the same ICN. [16]

4.3 Exception Handling

Exception handling is based on extending the ICN model by a “send to” operator and an extension to the “dispatch” and “schedule” operator. The “send to” operator allows a user to send a token to an *address* and to optionally provide a return address. An address is a 3-tuple consisting of an actor, a role, and an activity. Any of these three components can be left unspecified; a component need to not be within the same ICN. The “dispatch” operator is extended by an address resolution mechanism that guarantees that either nothing is specified or at least an actor is specified. If nothing is specified the “send to” operator has no effect. If no actor is specified but either a role or an activity or both, the “dispatch” operator first tries to determine the actor through the role and then through the activity. For example, if a token is sent to (Actor: “Not Specified”, Role: Manager, Activity: Order Entry), the operator finds an actor with a manager role and not an actor with a role of entering orders. If an address does not specify an activity but either an actor or a role or both, the token will assume an actor as location. These are “exception” states which represent another extension of the basic ICN model.

Each token maintains a stack of addresses. A “send to” operator pushes a new address on this stack. If the operator also specifies a return address, the return address is pushed on the stack prior to the new address. In any case, the “send to” operator sets the token state to “completed”. This activates the “schedule” operator which is extended such that it checks whether the address stack is empty or not. If the address stack is not empty it pops the stack and applies the extended “dispatch” operation on the popped address. If the stack is empty the “schedule” operator resumes normal operation. Thus, it is possible that *A* sends a token to *B* with return address to *A* and *B* sends the token to *C* without return address. By the time the token is completed at *C* the return address to *A* is on top of the stack so that the “schedule” operator sends the token back to *A*.

We found that the mechanisms to support dynamic change and the mechanisms to support exception handling are complementary: the exception handling features of Chautauqua allow users to "pick up" "fall out" tokens which result from a dynamic change; the dynamic change support of Chautauqua allows the adaptation of a workflow model to actual work patterns in order to reduce the overhead of exception handling.

5. The Chautauqua Implementation

Chautauqua provides tools for concurrent enactment of a model (including its change) including a graph editor for model editing and an extended World-wide Web server for document processing and other information presentation

We implemented Chautauqua as a client/server application. The server is a very general *active* object server. An active object server allows clients to register notification requests. The object server then notifies a client each time a commit meets the condition of a notification request. The notification requests are processed in the order of registration.

5.1 The Active Object Server

All state in the Chautauqua system is maintained in an *active* object server. The object server accepts requests from clients and returns a list of objects upon each request. However, the object server is active, i.e., it can also send objects to clients upon certain conditions within the server. Each client can register *notification requests* which define these conditions and associate them with that client. Whenever a client commits objects to the server the server checks all notification requests against the set of committed objects in the order of registration. A notification request matches if a subset of committed objects match the request's condition. That subset is then sent to the notification requestor along with information about who committed these objects. A special case is objects that are committed in order to delete them. Notifications do not exclude deleted objects so that a client can find out about object deletions.

The active object server is implemented in Python which is an object oriented, interpreted language. The server manages Python objects which are associative arrays with fast access methods and meta-information about their internal structure. This allows for very powerful and flexible operations on objects and their class definitions. For instance it is possible to add new attributes to objects on the fly without affecting other objects. Object class definitions are themselves Python objects and can be manipulated on the fly.

Clients of the active object server use a network interface in order to connect and interact with the server. The network interface allows clients to query the server, to register notification requests, to register new objects, to lock objects, and to receive notifications. The implementation of the network interface supports object caching and consistent referencing of persistent objects[1].

The client can choose between two ways of receiving notifications: the *notification by interrupt* installs an interrupt handler and spawns a child process that listens to the server for notifications. When the child process receives a notification it forwards the notification to the parent via a pipe and sends a signal to the parent process. The parent process picks up and processes the notification within its interrupt handler. The *notification by pipe* spawns a child process which connects to the parent process by a pipe that is provided by the client. Notification by interrupt is useful for clients that are otherwise not event driven. Notification by pipe is advantageous for clients that already have an event handler such as an application with a graphical user interface and where the read side of the pipe can be easily integrated into the event handler.

The query language provided by the server allows selection of objects based on object class names and on attribute/value comparisons. Values can be either primitive values and their sets or persistent objects and their sets. Comparisons include equality and inequality of values and comparisons of sets with values or with sets. The selection of objects is executed on the server side. Common database operations such as projection and join need to be implemented by the client and are currently not supported by the network interface.

5.2 The Token Mover

The token mover schedules tokens to new locations, merges them, and dispatches them to actors. It thus implements the "schedule", "merge", and "dispatch" operators. The server notifies the token mover whenever a client commits a set of tokens that are in the "completed" state. The token mover first applies the "schedule" operator and then the "dispatch" operator. It also maintains a queue of blocked tokens at each conjunctive join. The "merge" operator is applied when a correct number of matching tokens are located at a conjunctive join. The token mover also implements the "fall out" operator: the server notifies the token mover

1. This consistency management does not cover pointer variables which are local to the client application and not part of the network interface.

whenever a structural change occurs upon which the token mover queries the server for all tokens that are located at deleted locations. The token mover applies the “fall out” operator to the result of this query.

5.3 The Graph Editor

The graph editor implements the “edit” operator and is designed to manipulate the structure of an ICN while it is enacted by multiple workcases. After starting the editor, a user connects to the active object server and enters a name of a procedure to be edited. The editor displays the graph of this procedure and all relevant workcases in the viewer window. Each workcase is represented by a set of labeled tokens which are positioned at various locations of the procedure structure. Each token label displays the name of the workcase requestor, the workcase identification number, and the current token status. As work progresses these tokens move on to different locations. The viewer window is continually updated by the active object server making work progress and any structural changes immediately visible.

A user starts an editing session with an editor window which copies the current structure of the procedure displayed in the viewer. The editor allows one user at a time to change the procedure structure. During the editing session the user can still watch the progress of work in the viewer window. The user ends an editor session with committing the structural changes to the server which also closes the editor window. At this point the structural changes immediately take effect in the enactment of the procedure and are displayed on the viewer window. The viewer window helps the user to select an appropriate time to commit structural changes. If a structural change caused “fall out” tokens the viewer displays these in the bottom left corner.

5.4 The Web Server Extensions

We decided to use *Word-Wide Web* (WWW) technology for implementing forms and various tables in Chautauqua. The *HyperText Markup Language* (HTML) lends itself nicely for defining tables and forms, and modern World-Wide Web browsers implement sophisticated table layout algorithms and navigation functions. We generate all HTML code on the fly using a standard WWW server and Python scripts which conform to the *Common Gateway Interface* (CGI) standard (from now on called CGI programs). The CGI standard enables a WWW server to execute and uniformly communicate with programs which conform to the same standard. The HTML code allows easy specification of forms, tables, and *hyperlinks* to other HTML documents or CGI programs. A hyperlink is a *Unified Resource Locator* (URL) associated with a certain area in the display of the browser. An URL contains a communication protocol name, the server address of a WWW server, a document or CGI program name and other parameters. A user can follow hyperlinks by mouse-clicking on the associated display area which effectively either retrieves the document or executes the CGI program specified by the corresponding URL.

The WWW user interface of Chautauqua allows users to use any WWW browser that supports forms and tables. In order to connect to the Chautauqua system the user opens a certain URL which specifies the server address of the WWW server, the CGI program to be executed, and arguments to be passed along to the CGI program. These arguments in turn include at least the Chautauqua server address and the user ID of the user. The browser connects to the specified WWW server and sends the URL as a request using the *HyperText Transfer Protocol* (HTTP). The server parses the request and executes the specified CGI program with the specified arguments. If the browser issued the request in the context of a form, the server makes the form entries available to the CGI program as standard input. The CGI program connects to the Chautauqua server, performs read and write operations, and outputs HTML code. The server forwards this output back to the client as response to the client’s request and terminates the connection to the browser.

It is important to note that this architecture is very inefficient. We chose this design because of its fast and easy implementation. A much better architecture would be to integrate the WWW server and its extensions into the Chautauqua server. This would result in huge performance gains because it would obliterate the expensive serialization and deserialization of Python objects (about 80% of the retrieval overhead) and the expensive start-up times of interpreted CGI programs (sometimes more than one second in the case of Python scripts).

We will now turn to the various components of the Chautauqua WWW user interface:

5.4.1. To-do lists. The basic notion of the Chautauqua WWW user interface are *to-do lists* which are tables that show tokens sorted by actor, role, and activity. In the table, actors are called *employees* and tokens are called *tasks*. Tasks are labelled with their status, requestor, time of request, and urgency. A global to-do list lists the tasks of all actors while a local to-do list is specific to an employee. Each task contains a hyperlink to a form which is another dynamically generated HTML page (see below). Each to-do list corresponds to one ICN. The table entry that corresponds to the start activity of the ICN contains a hyperlink to a special form with which the user can create a new workcase. At the end of a to-do list is a list of blocked tokens. Hyperlinks lead to the corresponding forms.

5.4.2. Forms. Forms are rendering token and workcase attributes and are dynamically generated whenever a user clicks on a task in a to-do list. They are divided into six sections:

1. The *workcase section* displays information about the workcase of the token.

2. The *comment section* shows notes of the actor who completed the previous activity on this token, and allows the current actor to add notes.

3. The *form section* is a list of labeled entry fields. If an entry field has conflicting values, the corresponding entry field is empty and a menu with the conflicting values is displayed next to the entry field. The default value of the menu is “defer” which means that conflict resolution is deferred by default. The user can resolve a conflict by either choosing one of the conflicting values from the menu or by entering a new value into the entry field.

4. The *routing section* allows the user to create a decision list or specify a “send to” address. The address can be composed by selecting from three menus, the actor, role, and activity menu. Each menu contains a “not specified” item additionally to the corresponding items of the ICN. The user can also hit a “return” button which adds the return address to the address stack of the token.

5. The *control section* lets the user suspend, complete, or perform a “send to” operation by hitting a corresponding button. If the token’s address stack is not empty the user cannot complete but return the token to the stack’s top address.

6. The *history section* contains a table representing the history list of the token. The table contains also a history of comments.

A special form is the workcase creation form which differs from the normal form by providing entry fields in the workcase section and by skipping the history section. It also provides a hyperlink to a form editor.

5.4.3. The form editor. The simple form editor allows the user to textually specify and modify the sample form of the current ICN. Chautauqua uses this sample form to generate the workcase creation form and to instantiate the new workcase and its initial token. The syntax of the form specification is a list of entries of the form [global:] <field name>, where each entry has to be on a separate line, and where <field name> can be any single line string. The optional “global:” denotes global fields. At the bottom of the form editor is a button that submits the form specification.

5.5 The Task Monitor

The task monitor is a textual alternative to the Chautauqua graph editor. It is called with the Chautauqua server address and the ID of an employee. It then displays a continually updated list of all tasks that are assigned to that employee and which have the token status “new”.

6. Summary and Conclusions

This paper has presented the motivation, features, and design/implementation of Chautauqua, an exploratory collaboration management system designed and implemented within the Collaboration Technology Research group (CTRG) at the University of Colorado. This system represents a tightly knit merger of workflow technology and groupware technology. This collaboration management system is organizationally aware; it maintains a representation of goals and groups and roles and processes. It allows, but does not insist upon a rigorous representation of procedures, control flow, and data flow. Novel features include flexible exception handling mechanisms, representation of inconsistent concurrently updated information, assistance for simultaneous group editing, and a powerful, verifiable dynamic change capability. All of these features are accessible to any and all users with appropriate access rights.

Chautauqua is an ongoing research project; thus, we continually strive to further understand the nature of collaboration, to continually add functionality, and to test and evaluate this functionality. This document, therefore, only captures a snapshot of Chautauqua at one instant in time. Chautauqua executes over the internet; it is also a public domain system which is freely available over the internet. Our current primary objective is to gain further usage experience with Chautauqua.

7. References

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

[2] Bair, J. "Methods for Success with New Workflow Systems," GroupWare'92, edited by D. Coleman, Morgan Kaufmann Publishers, San Mateo, Ca.

[3] Bentley, R., et.al., “Supporting Collaborative Information Sharing with the World Wide Web” in Proceedings of the 4th WWW Conference, December, 1995.

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

[5] Cook, C., “Office Streamlining Using the ICN Model and Methodology,” Proceedings of the 1980 National Computer Conference. June, 1980.

[6] Collaboration Technology Research Group home page, Available on the World Wide Web at *http://www.cs.colorado.edu/~skip/ctrg.html*

[7] Ellis, C. A. and G. J. Nutt, “Office Information Systems and Computer Science,” *ACM Computer Surveys*, Vol. 12, No. 1 (March, 1980).

[8] Ellis, C. "OfficeTalk-D, An Experimental Office Information System" Proceedings of the First ACM Conference on Office Information System, June 1982.

[9] Ellis, C. A., S. J. Gibbs, and G. L. Rein, “Groupware: Some Issues and Experiences,” Communications of the ACM, Vol. 34, No. 1, January, 1991

[10] Ellis, C., and G. L. Rein, "rIBIS: A Real Time Group Hypertext System" in the International Journal of Man Machine Systems, 34, 1991

[11] Ellis, C. A. and G. J. Nutt, “Modelling and Enactment of Workflow Systems,” Proceedings of the 14tn International Conference on Applications and Theory of Petri Nets (June, 1993).

[12] Ellis, C. A. and G. J. Nutt, “Multi-Dimensional Workflow,” Proceedings of the 2nd IDPT Conference. December 1996.

[13] Flores, F., et.al. “Design of Systems for Organizational Communication” ACM Transactions on Office Information Systems 6,2, April 1988.

[14] Grudin, J. “Why CSCW Applications Fail,” Proceedings of the ACM CSCW88 Conference. August 1988.

[15] Johansen, R., and Swigart, R. Upsizing the Individual in the Downsized Organization. Addison-Wesley, 1994.

[16] Keddara, K., Ellis, C., Rozenberg, G. “Dynamic Change within Workflow Systems”, Proceedings of the ACM SIGOIS Conference on Organizational Computing Systems, Milpitas, CA, August 1995

[17] Kreifelts, T. "Coordination of Distributed Work: From Office Procedures to Customizable Activities," Verteilte Kunstliche Intelligenz und Kooperatives Arbeiten, 4. Internationaler GI-Kongress Wisensbasierte Systeme, Munchen, Germany, Oct., 1991, pp. 148

[[18] Mohan, C. “State of the Art in Workflow Management”, presented at the 5th International Conference on Extending Database Technology, March 1996.

[19] Nutt, G. J. and C. A. Ellis, “Backtalk: An Office Environment Simulator,” ICC ‘79 Conference Record, June, 1979, pp. 22.3.1-22.3.5.

[20] Saastamoinen, H.T., *On the Handling of Exceptions in Information Systems*, University of Jyvaskyla PhD Thesis Dissertation, November 1995

[21] Sheth, A., et. al. “An Overview of Workflow Management” in Distributed and Parallel Databases, 3,2. April 1995.

[22] Wainer, J. and C. Ellis, “Goal Based Groupware” in Collaborative Computing Journal, Vol.1, No. 1.

[23] “Workflow Management Specification Glossary” by the Workflow Management Coalition 1995.

[24] White, T. and Fisher, L. *The Workflow Paradigm - The Impact of Information Technology on Business Process Reengineering*. Future Strategies, Inc., Alameda, CA, 1994.

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

.

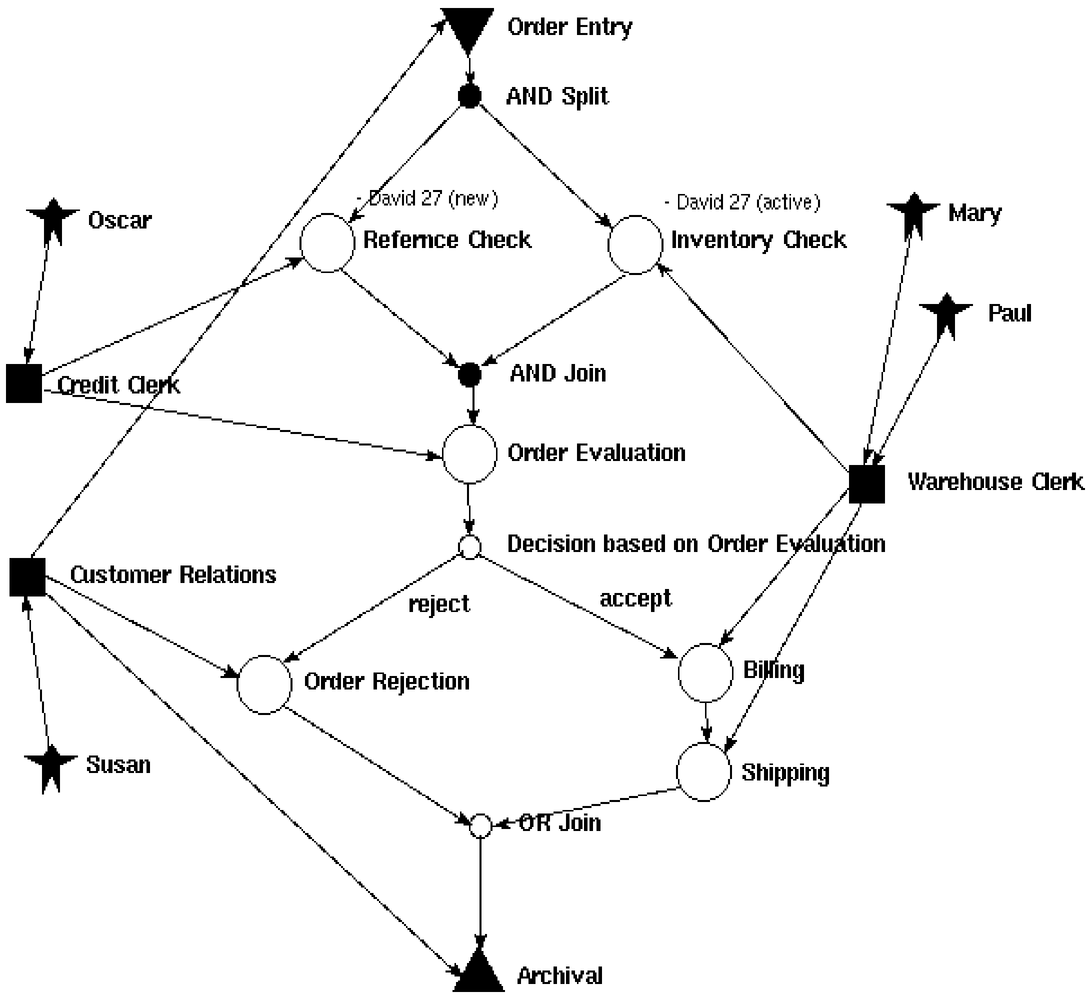

Figure 1. An Example ICN representing Order Processing.

. .

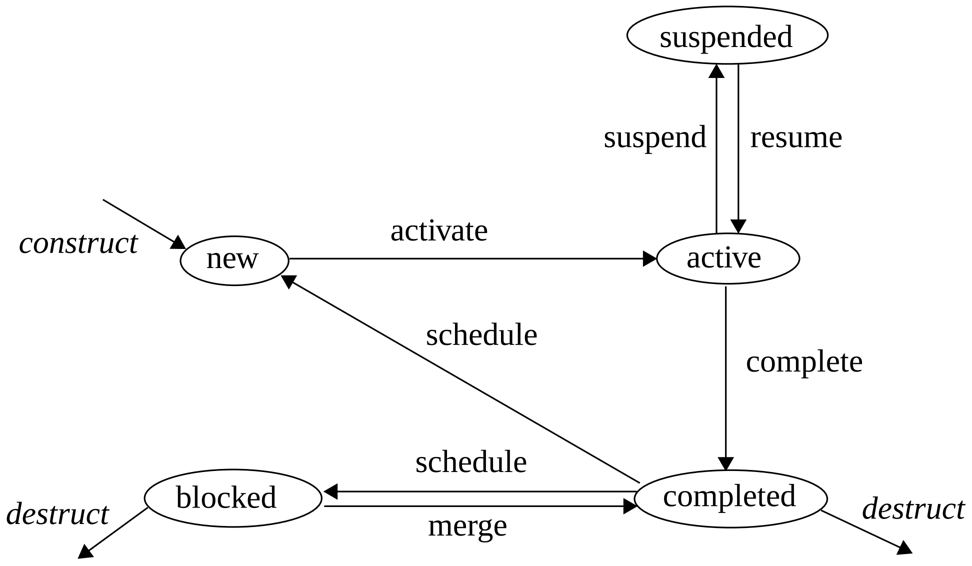

Figure 2. The token status transition diagram. The events in italics are constructors and destructors of tokens. All other events are enactment operators.

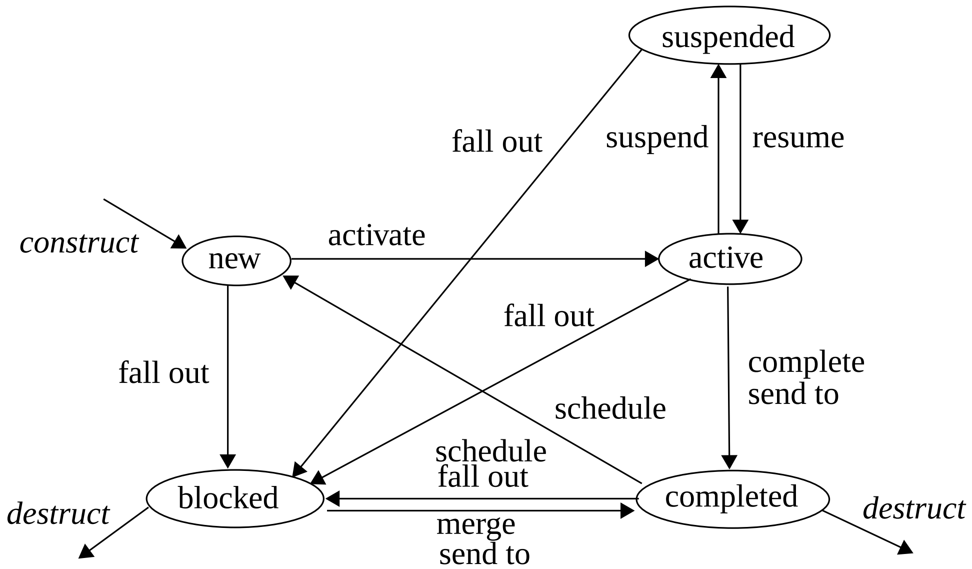

Figure 3. The extended token status transition diagram.