INTERNATIONAL JOURNAL OF INTELLIGENT CONTROL AND SYSTEMS
VOL. 12, NO. 1, MARCH 2007, 63-71

CaseFlow: A Prototype Rule-Based System to Support Case-Based Work

Paul R. AUSTIN, Tao LIANG, Patricia WALL, Scott WEBER

Abstract — **A rule-based prototype is proposed as an alternative to traditional workflow applications for case-based business processes, where a "case" is a collection of documents representing the relationship between a business and a customer in support of critical business decisions. This project is based on ethnographic studies of office work where we have observed the organization of complex processes as cases in a variety of settings. Cases reflect the language used by office workers and provide a very common metaphor for organization of information. The information content in a case is diversely represented and largely unstructured, making it a challenge for electronic processing by conventional data processing systems. To address these issues, we have developed a prototype system called CaseFlow that uses a rule-based workflow engine and organizes information around the concept of a case.**

Index Term **– Ethnography, Work Practice, Cases, Rule-based, Workflow, DocuShare**

INTRODUCTION

This paper describes a rule-based prototype that offers some interesting advantages over many current workflow systems for certain kinds of case-based work. Called "CaseFlow", the core idea shifts the focus from the "process" to the "case". Case-based work is quite common in many governmental, educational, contractual, legal, and medical contexts. Within an organization, a case is an information classification concept, used to inform one or more important decisions. Cases seem a more natural description for certain kinds of workflow work than sequences, flowcharts, or process graphs. Case workers have a good understanding of their own role and perhaps adjacent roles, but seldom of the whole system.

Our primary objective is to develop a flexible approach to case-oriented processing that is easy to understand and use, while still accommodating the many exceptions and variations that occur in the real world. Our tests for ease of use are quite simple: can a customer create and maintain their business rules with little or no professional IT assistance? Does use of the system facilitate rather than hinder business processes, faithfully enforce policy, adapt easily to changing business environments, and build on the documents and information that the customer already uses?

Our preferred method is to work as a cross-disciplinary team with collaboration between ethnographers, software developers, and a business that organizes some aspects of their work around cases.

We employ a combination of generally well-known technologies that in the aggregate create an interesting solution.

The next section describes our work practice methods through which we study customer work practices, inform software system development, and evaluate the effectiveness of the developed systems. Following that, we describe some of the limitations of the current state of the art in workflow systems and the technologies used to overcome some of these workflow limitations. A later section describes the prototype system as developed. We continue to find opportunities for further research as well as different business contexts for application of the system.

Paul Austin, Pat Wall, and Scott Weber are with the Xerox Innovation Group, 800 Phillips Road, Webster, NY 14580, email paul.austin@xeroxlabs.com, pat.wall@xeroxlabs.com, and scott.weber@xeroxlabs.com.
Tao Liang is with the DocuShare Business Unit, 3400 Hillview Avenue, Palo Alto, California 94304 USA, email Tao.Liang@xerox.com

WORK PRACTICE METHODS AND OBSERVATIONS

It is part of our design practice to inform and guide technology and research efforts based on studies of real business work. We do this by conducting ethnographic studies of customer work place contexts in order to understand their current work practices, identify issues with those practices, and discover opportunities for solutions and technologies that will support their work. Originating in the social sciences, ethnographic methods were originally used to study cultures. In recent years ethnographic methods have been applied to study work activities in business contexts to inform product, services and systems development [8], [9], [10], [12], [13]. Figure 1 illustrates of the work practice methodology in use in our organization. Ethnographers conduct open-ended interviews and observations of people as they perform their everyday work activities in order to develop in-depth understandings of what it takes to accomplish the work. These understandings are used to help guide and inform the development of technologies that will fit in with and support current organizational practices. Our research has focused on the study of business contexts to characterize work practices in office environments.

Through recent ethnographic studies of key business processes and their associated document lifecycles we've observed that some business processes are organized and managed as cases. Case-based processes can be characterized as a collection of documents pertaining to an individual customer, organized and maintained by a business (or agency or organization), which constitutes the basis for governing the relationship between the customer and business. Each of the documents in a case is associated with a set of pre-defined procedures that involve multiple people or groups, perhaps including people external to the organization, and has specified handoffs from one stage to the next.

Cases form an effective model to support particular kinds

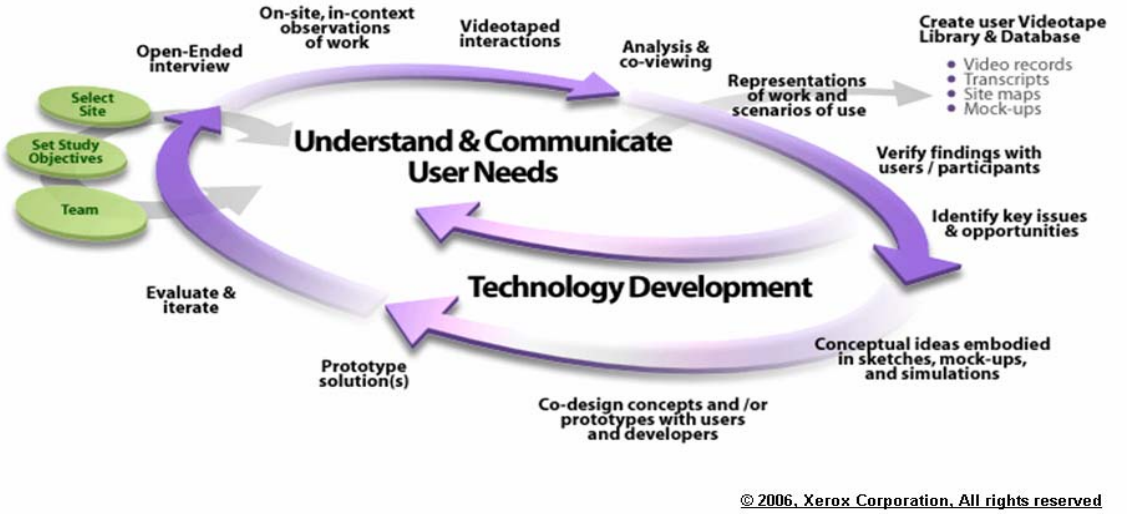


Figure 1: Work Practice Methodology

of business transactions. Some interactions between customer and business are handled immediately, and with little or no state or case history. Consider the example of a cash purchase in a store: such interactions are often well supported by a transactional database system, wherein workflow can be implemented through reports and queries. These interactions are often serialized, where the business concludes an interaction with one individual before moving on to the next. (Hence the dreaded "price check"). There is often little room for negotiation or interpretation; the price is this, the customer has or does not have the required payment, so the transaction is completed or dropped. Transactional systems effectively automate a large number of these workflows already. These systems impose rigid rules by design, at the expense of flexibility and changeability in the system. Exceptions to the automated process, while necessary, are more complex.

In contrast, we will focus on cases that represent an on-going relationship between customer and business, where information is represented by a variety of documents which are characterized by largely manual processes in which agents of the business must acquire the needed information from the documents and make decisions in accordance with the business policies. Such cases introduce several problems that must be addressed for a workflow system to provide effective automation.

Examples of case organized work include:

- mortgage applications and lending decisions
- legal cases/contract development
- patent applications
- trial preparations
- crime and accident investigations
- insurance claims
- medical records
- government assistance applications,

It is evident that cases can constitute an interface between multiple parties who each handle portions of the information.

We will examine one of these examples, the mortgage application process, in more detail. The process involves the collection of information in several documents, all of which are assembled and used to make a decision about whether or not to grant a mortgage. Submission of a mortgage application triggers the initiation of a mortgage case for an applicant to a financial institution. This in turn triggers a series of workflows aimed at collecting appropriate information about the applicant's financial history and status in order to assess the risk of loaning funds. In the situations we've observed, these workflows are primarily manual activities, dependent on various people in the lending organization, as well as the customer, requesting information, from previous employers, credit bureaus, landlords, and letters of reference, etc. to complete the customer's financial profile. There is often a master document or checklist kept in the case file that is used to check the status of each required form of documentation. This enables the various people working on the case to check the case status and provide new updates as they become available. The checklist is also a prompt to follow-up on information that has not been received. One issue associated with the paper-based case file is that it is not easily shared simultaneously among the various people working on the case. And although some institutions are migrating to electronic mortgage processing, this is not uniform across the industry. Paper documents are still part of the application process.

Another example of case-based workflows is embodied

in the creation of legal contracts. In one study we observed the contract development process between a corporation and vendors supplying services to the corporation. This work includes several iterations and exchanges of contracts, often shared and communicated via email and phone conversations. The attorneys made extensive use of email as a personal repository/archiving system for these contracts. The use of email to store/archive and organize work has been noted in other studies ([14], [11]

DESIGN PRINCIPLES

Our studies of office work highlight an opportunity to explore the application of automated tools to support activities in the context of case-based processes. In order to develop a prototype technology to support case-based work, several design principles need to be considered. The technology must:

- Make it easy for those using the system to access, navigate and use the data and workflow
- Support the use of unstructured (human readable) data and information in documents that are integral to the work
- Present multiple views and access to the data across concurrent cases (overview) and within cases (detailed view)
- Enable multiple users to access workflows and data simultaneously.
- Enable user control, flexibility and the ability to modify workflows to accommodate exceptions and changes to improve the workflow
- Build on existing workflow tools and metaphors already in use, such as email and enabling use of electronic forms similar to paper-based forms already in use, expand
- Support blend of manual and automated tasks as needed within specific workflows.
- Insure the case keeps progressing by prompting and reminding users as critical dates approach.

Challenges Characterizing Case-based Work

From a technical perspective, a case is a collection of information; in practice this information arrives unpredictably, is unstructured, incomplete, fraught with exceptions, concurrent with many other cases, and directed toward human rather than machine processing. These and related characteristics can make application of traditional workflows to cases challenging.

We will use the term "workflow" to characterize many popular and readily available systems that use a graph or flowchart to indicate the processing paths for a business. Often these systems support a graphical interface through which the designer can drag and drop various predefined processing elements and draw interconnecting lines to indicate flow of control. Several systems further use the graphical view to indicate the status of a particular case by highlighting a node in the graph to represent state, and some can aggregate all workflows together on a graph to indicate processing backlogs. While constituting a fairly intuitive interface for the designer, especially in the rare case where a process is documented similarly, these workflow graphs are somewhat inflexible at handling alternative sequences of events, exceptions, and changes that tend to have widespread implications.

Representing business rules as code, or even flowcharts or graphs, is often too difficult for office workers who are primarily focused on getting their work done. This representation also requires a very complete knowledge of all related business processes, and tends to focus attention on the "normal" (unexceptional) case. When exceptions are considered, they often impact a substantial area of the code or graph, making them both more difficult to represent and more likely to disrupt something else.

"Embedding business rules in applications compromises the versatility of the application and consequently the business process that it supports. Embedding obscures the semantics of the rules. Embedding creates unwanted co-dependencies with procedures and reduces independence between rules. Embedding makes rule changes much more difficult." [1]

Cases are concurrent.

The on-going nature of relationships between businesses and their customers imposes a high degree of concurrency on the business processes. The business is typically involved with many customers at once, with their cases in various states of completion. New customers and new information for existing cases arrive concurrently via various media. The business must process the information in a timely manner, and communicate its decisions back to the customers. Using computers to monitor the status of cases and prompt people to provide needed information can be very helpful; businesses are often dealing with more cases concurrently that can be individually remembered and sometimes cases stall unnecessarily due to interruptions by other cases.

When a sequential flow chart or workflow sequence is imposed on this information several problems can occur: information that arrives before it is needed in the sequence may be lost and require additional work and delay for re-submission, or may lead to the assumption that the case is more advanced than merited due to missing information, or sequences may have to be expanded to anticipate all arrival sequences.

Workflow understanding is distributed.

Business responsibilities are often distributed over many people working in several different roles. This introduces communication challenges when case decisions must be based on the input from several people. While people serving particular roles are usually quite specific about the work that they do, few are aware of all of the roles of all of the people involved in the aggregate decision. This requires

that developers perform "knowledge engineering" activities through which information from many people is collected and integrated to form an accurate overall understanding of the business processes. As new knowledge is required, the draft workflow must be modified to accommodate new exceptions and processing. In a conventional workflow this can sometimes have wide ranging implications.

Exceptions are normal.

Businesses operate in a very complex environment, exposed to many factors which they cannot control, even more as businesses outsource more services. People can take vacation days, make mistakes, or change their minds. An effective workflow system must accommodate countless exceptions easily, or potential productivity benefits will be lost trying to use an inflexible system. For example, a vacation by a decision maker may stall progress in a conventional workflow system even though a higher authority is willing to override the normal sequence. Resuming progress often requires specialized intervention that is rarely used and hence difficult to remember.

Information is unstructured.

Information is represented in many forms, not all of which are electronic. Even among electronic representations, consistent annotations (such as those based on XML schemas) are just beginning to emerge, and information exchange between electronic forms is still often a manual process. An effective workflow system must facilitate the collection and conversion of information to a form by which the case decisions can be made, often by incorporating manual processes with automated ones. Information is often manually extracted, filtering out irrelevant portions, and used in formulating decisions about the case. Unstructured information may not fit well into a database because of differences in ordering, description, units of measure, and many other factors. These documents were generally designed for human consumption, relying on an intelligent reader to interpret the necessary information. Machine understanding techniques are making some promising advances but were felt to be too immature during our prototype development. We provide an interface through which users extract information from an unstructured document and provide it in a format that facilitates automated processing. We store this as metadata for a document.

Workflow systems become particularly problematic when a mistake is made in manual extraction; how does one determine the state to which the workflow must be rolled back, particularly if subsequent work has been completed? The workflow may not even have a suitable rollback state.

Change is constant.

Since the business environment is very dynamic, change is a constant. While some changes are easily accommodated by perhaps adding a step in the correct workflow sequence, other changes can cut across the workflow with broad implications. Consider a policy change that requires all approvals be digitally signed; virtually every approval in the workflow graph would require modification. The role of the CaseFlow system is to automate the collection and organization of the information to facilitate the decision makers.

Rule-based Processing has advantages for case management.

In a rule-based system, processing is controlled by a set of rules, each of which specifies a set of conditions for which the rule applies and a set of actions which are to be taken when the conditions are met. Processing is achieved by choosing one or more rules that apply to a given initial condition and then executing the actions of the selected rules, which may change the initial condition and then recursively permit selection and execution of additional rules. There are many schemes for efficient selection and execution of rules that have similar semantics described in the artificial intelligence literature. Broadly speaking, these schemes fall into "forward-chaining" and "backward-chaining" categories. In forward-chaining we attempt to execute all rules that apply to a condition until no additional rules match the *modified* condition. In backward chaining we attempt to find a sequence of rules that, when applied to the initial condition, result in a desired specified goal state. Both approaches can be used efficiently, and some systems use both together. At present we are using a forward chaining approach, where the information in a case triggers execution of rules that interact with users or adds information to that same case.

In using rules we seek to be highly declarative: each rule should be entirely self-contained and represent a relationship that is true in all situations that meet the rule's conditions. Rules written this way are largely independent of each other, and hence have a significant modularity advantage in that they may be defined and executed independently. The rule author's focus is on defining a truth relationship where the actions are always required when the rule's conditions are satisfied. Hence, the conditions of a rule completely define its applicability; there is no additional "context" information that must hold for a rule to be fired. This property allows rules to be changed independently of each other without the concern that the change will cause unintended consequences in other parts of the system.

In contrast, programs, flowcharts, and workflow graphs rely heavily on context represented as a position in the program or a location in the graph. This dependency is anti-modular and spreads the impact of any change made to the graph. The designer is responsible for the impact of any change on all portions which are dependent on the context in which the change is made, frequently without any explicit indication of the elements that are dependent. However, context can reduce the number of conditions evaluated. [1] observes this may be a legacy from an era of limited and precious computer resources rather than today's largely idle

processors.

Another advantage of rule independence is that rules handle exceptions gracefully; either inherently or simply by adding a new rule that handles the exception. Consider the following example comparing a traditional workflow with a rule-based one: Steve wants to purchase some software that is expensive, so he needs to get second level management approval from Chris before he can place the order. Normally, Steve sends his request to his manager, Anna, who decides whether to forward it to Chris. But Anna is on vacation for two weeks and Steve is anxious to get the software soon, so he talks to Chris and gets his approval. In a simplistic workflow application the request would stall at Anna's approval level until she returned and made her decision. To override this Chris would have to manually intervene and emulate Anna's approval. In the rule based scenario, one rule would represent that "if Chris approves Steve's request, and Chris is Steve's second level manager, then Steve may place the order". Hence, Chris's approval is sufficient without emulating Anna's decision. This also leaves a more accurate record of the approval.

Rules seem to relate well with workers. Often we find that individual workers can describe what they do in accurate detail, and can be prompted to describe exceptions when we ask "what-if" questions, but do not understand the entire process in detail. Once again, rules facilitate capturing each worker's understanding somewhat independently, rather than having to build up a more elaborate integrated view of the entire process.

Rules handle concurrency well. The information needed for a case decision tends to arrive in an arbitrary order. Since rules are triggered when their condition is satisfied, it is sufficient to simply specify the multiple information items needed as a conjunction. In contrast, traditional workflows grow dramatically when trying to capture arbitrary order of arrival, since a distinct branch is needed for each possible order, along with appropriate "fork" and "join" operations.

Rule based systems are potentially of exponential computational complexity; however for the modest number of rules and conditions that we anticipate in most business process workflows we do not expect to experience lengthy computation times.

Other examples offering rule-based workflow solutions, some with case concepts are found in [1], [3], [5], [6], [7]

DESCRIPTION OF THE PROTOTYPE CASEFLOW SYSTEM

We developed a working prototype of the CaseFlow framework to explore the possibilities of rule-based workflow. We then developed a prototype around the concept of mortgage application processing on the framework, for which we had work practice study data. The prototype application is not complete as we have not implemented all of the steps in the process. However, we have implemented enough to understand the advantages of a rules based approach.

Technologies Used in Prototype

The prototype framework was developed using Xerox' DocuShare document management system as the repository. A simple rule processing engine was developed using Java. The rules and data are represented using XML, and the XML is converted to HTML for viewing by the users.

DocuShare

DocuShare from Xerox is an easy to use document management system that is readily extended into new capabilities. We have used it as a basis to develop specific applications for NASA, DOD, and other government agencies, as well as commercial and educational organizations. We believe that it forms a sound platform for representation of cases and rule-based workflow.

We built our CaseFlow example by exploiting a number of DocuShare's features: out of the box DocuShare is capable of representing a rich variety of documents as well as meta-data about the documents, such as their source and privileges for access and change. DocuShare may be extended by adding new object types and relationships between objects, and is capable of representing ontologies. DocuShare provides a robust and flexible security mechanism. In addition to its web-based interface, an email interface is provided. Internally, DocuShare is capable of delivering events to the CaseFlow framework when any change to the repository occurs. While other repositories could clearly be used, the rich features of DocuShare saved us a considerable amount of time in implementing CaseFlow.

XML / XPath

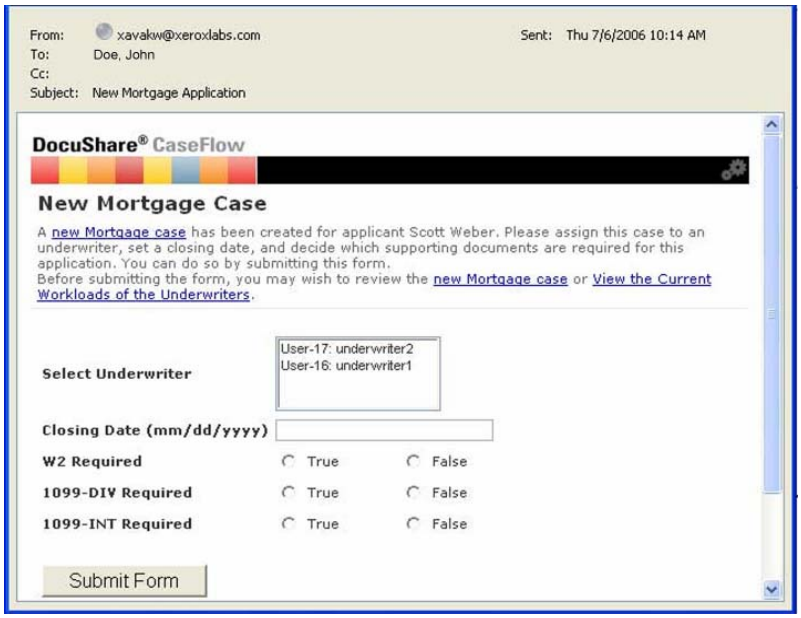
XML constitutes a flexible *syntax* for representing many kinds of information, and together with a schema or data-type-definition, constitutes a *language* that may easily represent domain specific (or vertical application specific) information. XPath is a query language that can select some elements of an XML document matching context (the element path) and/or content (text contained in elements). XPath can also represent arbitrary logical combinations of XPath queries, and thus is capable of representing the condition portion of a rule. XML is also suitable as a language for representing the actions of rules where such actions include modifying the contents or metadata of a case, interacting with information workers or clients, or initiating a conventional workflow.

Use of XPath does not require a schema or DTD, so we have not imposed such a requirement in CaseFlow. While this maximizes the flexibility for representing case information, rule authors must be aware of the expected representation and must take into account any variances.

Email Forms/XForms

Since many aspects of case work involve collecting information from people who are the subjects, managers, or

decision makers related to the case, we have made form processing an integral part of the case processing. Rules may



©2006 Xerox Corp. All rights reserved.

Figure 2: Email Screen Capture

trigger delivery of suitable forms to any of the case participants, and receipt of an electronic form can trigger additional rules.

From our work practice studies we find that email plays an important role in many information workers' daily routine. To accommodate this we permit delivery of a form within an email message (using HTML), which is submitted back to the CaseFlow system by emailing the result to the DocuShare server. This permits the worker to resolve the issue quickly and conveniently without the requirement to be online, access a web site, or cut and paste information, etc. Offline access was an important consideration in several of the scenarios we studied; email-based forms allowed us to address this need.

DocuShare's email interface permits use of certificates to achieve a high degree of security for email messages through encryption, and a similarly high degree of confidence in authentication through digital signatures.

XForms is a technology that extends HTML forms by separating the model from the visual representation and providing for constraint evaluation at the form client. In the future we intend to employ this in our forms, but it is not yet sufficiently deployed in web browsers and email programs.

Rules Engine

A simple rules engine for our prototype was implemented in Java. This engine evaluates the rules in the sequence they are listed in the rules definition file. This simple rules engine was robust enough for our prototyping efforts; however, in a production environment a more complete/commercial engine might be desirable. The documents and metadata are exposed to the rules engine as an XML document. The condition portion of each rule, expressed as an XPath query, is evaluated against the XML view of the documents and metadata. Those rules for which the XPath expression returns a Boolean true value, the designated rule actions are performed.

Prototype Architecture

The prototype was constructed using the Java programming language and XML for data exchange between the interacting systems. This architecture is both robust and simple. The largest gap in the current development is a smart user interface permitting users to easily define and edit the rules for a case application.

Cases as Collections

The object model within DocuShare is extensible. Any of the basic object types in the system can be sub-classed, in the traditional Object Oriented Programming model, into a more specific object and can have specific meta-data associated with it. DocuShare uses the concept of collections for grouping together different objects within the repository. We created a specific collection subtype called a case. This was a natural extension of what we saw in fieldwork, where workers have paper files associated with each case. The case subtype supports additional metadata for the case-based processing. Subtypes of "case" can be defined in turn, with additional metadata specific to mortgages, insurance claims, or other sorts of cases. We also created document sub-types for the specific types of files encountered in each case, again with additional metadata. This constitutes a virtual representation of the artifacts that workers traditionally managed as physical case files.

The event notification capability of DocuShare was used to trigger the Rules Engine to evaluate the rules of a particular case type. When a change is made to a case collection, or to any documents contained within it, DocuShare issues an event to all listeners. The rules engine registers as one of the listeners. On receipt of a change event, it evaluates the rules for that case type. If a rule condition evaluates to true, it will trigger actions that may further change the state of the case in DocuShare and recurse through the rule evaluations again.

XML Throughout

CaseFlow is largely based on XML. The case, as it is represented in DocuShare, is extracted as XML from the server for use by the rules engine after each state change. The rule conditions are XML XPath expressions, which evaluate to Boolean values. The available reports for tracking the status of cases are XML with appropriate XSLT transforms for viewing. XML affords the possibility of plugging other components and (web) services into the

system. Through the separation of information and presentation, we gain the flexibility to deliver UI elements to a variety of devices from web browsers to email to hand held PDA's.

Email / form interactions

A major piece of development of the CaseFlow framework was centered on the creation of an HTML email form capability for user interaction with the system. Email has become a standard way for knowledge workers to interact, and the typical knowledge worker spends a large amount of time in his email client each day processing email. Typically, email would be used to send a worker an alert or a message that there is some work waiting for him. There may or may not be a link to the system where the work is waiting, depending on whether the other system is web-based. If there is no link, the worker would then need to exit his email client and go to the new interface for the system to do his work. Our system sends to the worker both the alert and an interface into the system for him to do his work. This method has two advantages. First, the worker does not have to leave his email system to do his work. Second, by using email forms we allow the user to be able to do his work offline. For example, a user could be working remotely; he logs in to synchronize their email and then logs off to process it. As he is processing his email he can, via the HTML email, do his assigned work and then queue the response to be sent the next time he connects. We exploit the HTML presentation capability in Outlook (and other tools), and use "mailto" links to convey the information back to a system inbox. An issue with this method of interaction is that the propagation of viruses via email causes email tools to deliberately obstruct sending email from a form, requiring the user to approve the outgoing message. We have yet to discover a viable workaround for this.

Since email is not inherently secure, we require the use of both encryption and digital signatures with email; this prevents a hacker from masquerading as another user unless he has the **required private key.**

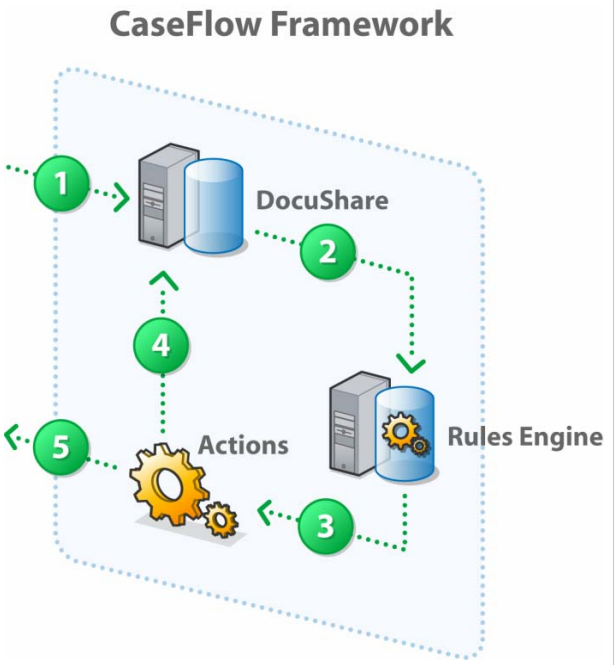
Figure 2 is a screen capture of the first HTML email sent to the loan administrator in our example case of mortgage application processing.

Architecture

The sequence of processing in the system is shown in Figure 3. Figure 4 outlines an example of the interactions between the CaseFlow framework and users. The prototype application built on the framework is the processing of a mortgage loan application.

CaseFlow Concepts in DocuShare CPX

Based on this work, some aspects of the CaseFlow framework are implemented in a new Xerox product, DocuShare CPX, aimed at supporting complex workflow processes. This first commercial version focuses on a core concept of CaseFlow – rule driven processes. Instead of building a custom rules engine, however, existing event and filtering mechanisms are used for rule selection and a traditional workflow system is used to evaluate the rules.



© 2006 Xerox Corp. All rights reserved.

Figure 3: Processing Sequence

1. New data is received by the DocuShare server which changes the state of a case.
2. The change in state causes DocuShare to issue a DocuShare event. The Rules Engine is listening for these events and begins evaluating the rules.
3. A rule has evaluated to a 'true' state. This causes the Rules Engine to initiate any actions that are associated with the rule.
4. An action can cause yet more state changes within the case on DocuShare.
5. Or, an action can send information back to the user. This information could be an email form or a web page.

In CPX, a rule is attached to a container object (a Collection) or a content object (a Document). Specified by end-users, a rule consists of three main components: 1) a triggering event; 2) a filtering condition; and 3) an action to be performed when 1 and 2 are satisfied. An example of a commonly created rule attached to a Collection: when a new Document is created in this Collection (event), and the title contains "paper submission" (condition), send an email to a specified address (action). In this version, users choose from a fixed set of events, conditions, and actions.

Since the action in a rule can produce an event that will in turn trigger the rule itself, measures need to be taken to prevent infinitely recursive rules. For this initial version of

CPX, a limit mechanism is included: a rule that has been triggered too many times within a specified period will be disabled and never be run until manually re-enabled. In future versions, more sophisticated mechanisms will be provided.

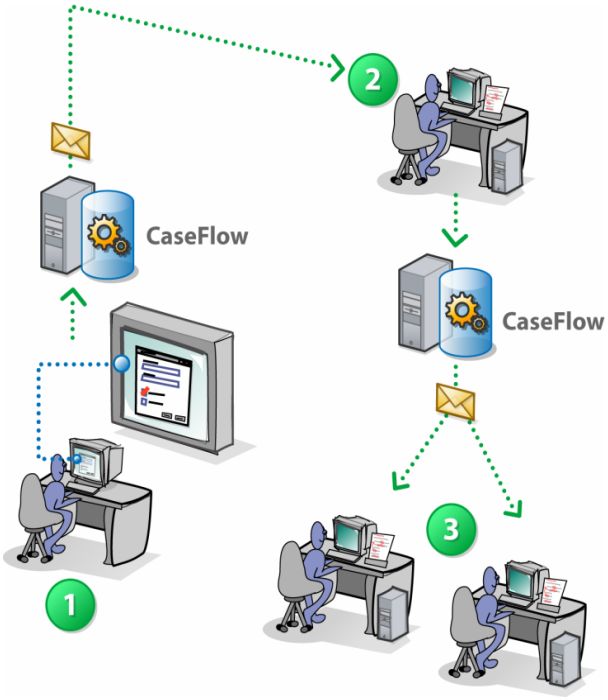


Figure 4: Mortgage Scenario

1. A user creates a new Mortgage Case on the DocuShare server. This triggers CaseFlow to begin processing the rules for a mortgage case.
2. CaseFlow sends an HTML form via email to the loan administrator, who fills out the form with the necessary data and sends the response back via email to the DocuShare server.
3. CaseFlow evaluates the data from the loan administrator and makes the changes to the state of the case in DocuShare. The change in state then causes more rules to be evaluated and additional emails are generated and sent to the loan underwriters and others concerned with the processing of the mortgage.

Other CaseFlow concepts are planned for future versions of CPX. Users will be able to specify multiple actions in a rule. Furthermore, the system will allow custom actions to be inserted into the rules system by a system administrator.

Next Step: Study of a Customer Deployment

The prototype has convinced us that there is significant potential in this approach, and has facilitated communicating the ideas to other groups that provide software and document solutions to a wide range of customers. Their evaluation is continuing and may result in a customer engagement that would permit us to study the concept in use, informing future development.

A deployment would help guide many interesting avenues of research. For example, we would like to understand the requirements for a user interface (UI) to allow users to edit the rules that govern a CaseFlow workflow. As explained earlier, our architecture uses XPath expressions to describe the rules. We would like to explore UI concepts that do not require the user to know XPath in order to edit the rules. The UI needs to be designed in such a way that an office worker would be able to change the functioning of the case. Another avenue of research is the continued exploration of email as a suitable and convenient interface to the CaseFlow system. Further interactions with customers and their case-based work will guide this research.

Acknowledgements

We would like to acknowledge the contributions of several others who have helped shape this work. Colleagues at Xerox Research Center Europe conducted field studies and documented some of the first observations of case-based work several years ago. Charlotte Baltus created the original implementation of the rules engine. Judy Slein was responsible for building the majority of the CaseFlow framework and developing the Mortgage application processing prototype. Jon Dorsey and Larry McKnight reworked our graphics.

References

[1] BizTalk Server Business Rules Framework (White Paper), Microsoft, December 2003, URL http://download.microsoft.com/download/e/6/f/e6fcf394-e03e-4e15-bd80-8c1c127e88e7/BTBusRul.doc

[2] Expert Systems , July 2006, URL http://en.wikipedia.org/wiki/Expert_system (a quick overview of expert and rule-based systems with references), also URL http://en.wikipedia.org/wiki/Rule-based_%28programming%29

[3] Rule-based Workflow (white paper), Verilét, Inc., 2005, URL http://www.verilet.com/pdfs/Workflow_WhitePaper_06282005.pdf

[4] Workflow Basics: Architecture and Applications, J. Leon Zhao, January 2002, SIG on Process Automation and Management, URL http://www.stevens-tech.edu/sigpam/publications/tutorial/HICSS-35/Workflow_Basics.pdf

[5] ARM Loan Audit Workflow, Visionetsystems Inc., 2004, URL http://www.visionetsystems.com/visionet/arm_law_casestudy.htm

[6] KANA Solution for Case Management Support, KANA Software Inc., 2006, URL http://www.kana.com/solutions.php?tid=46

[7] Workflow, HYLAND Software, 2006,URL http://www.hyland.com/English/Products/OnBaseProductModules/ManagementModules/Workflow, also URL http://www.onbase.com/English/IndustrySolutions/PublicSector/CaseStudies/Cuyahoga

[8] Anderson, W.L., Crocca, W. T. Engineering practice and codevelopment of product prototypes. *Communications of the ACM*, Vol 36, No 4 (June 1993), 49-56.

[9] Blomberg, J., Giacomi, J, Mosher, A., Swenton-Wall, P. Ethnographic Field Methods and Their Relation to Design. In *Participatory Design: Perspectives on Systems Design*, D. Schuler and A. Namioka (Eds.), 1993, Lawrence Erlbaum Associates, Hillsdale, NJ, 123-157.

[10] Brun-Cottan, F., Wall, P. Using Video to Re-Present the User, *Communcations of the ACM*, 38, 5, (1998), 61-71.

[11] Ducheneaut, N., Bellotti, V. E-mail as habitat: An Exploration of Embedded Personal Information Management. *Interactions*, Vol 8, No 5, (2001),. 30-38.

[12] Hughes, J, O'Brien, J, Rodden, T, Rouncefield, M, Blythin, S. Designing with Ethnography: A Presentation Framework for Design. In *Proceedings of the conference on Designing interactive systems: processes, practices, methods, and techniques*, Amsterdam, The Netherlands, 1997,, p 147-158

[13] Suchman, L. Office Procedure as Practical Action: Models of Work and System Design. *ACM Transactions on Office Information Systesms*, Vol. 1, No 3, October 1983,. 320-328.

[14] Whittaker, S., Sidner, C. (1996). Email overload:exploring personal information management of email. *Proceedings of CHI'96*. New York: ACM Press. 276-283.

[15] Functionality and Limitations of Current Workflow Management Systems, IEEE Expert, 1997, G. Alonso, D. Agrawal, a. El Abbadi, C. Mohan, URL http://www.almaden.ibm.com/cs/exotica/wfmsys.pdf

Paul Austin received the Ph.D. degree in computer engineering from Syracuse University and MS and BS degrees in electrical and computer engineering from Clarkson University. He is a principal member of the research staff at the Xerox Innovation Group. His research interests are in the areas of distributed systems, real-time control, ethnography, computer languages, algorithms, and artificial intelligence (or at least advanced knowledge representation).

Tao Liang received his Ph.D. from Stanford University and a B.S. degree from Drexel University. Tao is an engineering manager at the DocuShare Business Unit of Xerox Corporation. There, he manages a team of developers to develop new software products and services. He also works to commercialize emerging technologies from Xerox research centers and collaborate with external partners. Prior to this, he was a principal of new business development at the Internet Business Group. Prior to that, he was a research scientist in the Systems and Practices Laboratory of Palo Alto Research Center, Incorporated (formally Xerox PARC). There, he worked on numerous internet-based technologies to enable distributed collaboration, knowledge sharing, and digital rights management.

Patricia Wall is a research scientist and manager of the Work Practice and Technology group at Xerox. She received an MS degree in Psychology/Human Factors from North Carolina State University and a BS degree from Statue University of New York at Brockport. Her research interests include the application of ethnographic methods in services contexts and technology development.

Scott Weber received a BS degree in Mechanical Engineering from the University of Illinois Urbana-Champaign. He is a member of the research staff at the Xerox Innovation Group. His research interests are in the areas of ethnography, content/document management, computer languages, and the web.