Distributed Coordination and Workflow on the World Wide Web[*]

Antonietta Grasso, Jean-Luc Meunier, Daniele Pagani, Remo Pareschi

Rank Xerox Research Centre, Grenoble Laboratory
6, chemin de Maupertuis - 38240 Meylan - France
{grasso, meunier, pagani, pareschi}@grenoble.rxrc.xerox.com

ABSTRACT:
This paper describes WebFlow, an environment that supports distributed coordination services on the World Wide Web. WebFlow leverages the HTTP Web transport protocol and consists of a number of tools for the development of applications that require the coordination of *multiple, distributed* servers. Typical applications of WebFlow include distributed document workspaces, inter/intra-enterprise workflow, and electronic commerce. In this paper we describe the general WebFlow architecture for distributed coordination, and then focus on the environment for distributed workflow.

KEYWORDS:
Distributed Workflow, Coordination Technology, Collaborative Systems, World Wide Web.

1. WWW and the Inter/Intranet: the socio-economic opportunities

Teams, markets and hierarchies identify three different socio-economic relationships characterising modern organisational life. If we take the point of view of transaction costs economics (Williamson, 1975), these three different types of organizational arrangement lie on a continuous spectrum (Figure 1). At one end of the spectrum there is the pure market relationship, where the two actors engaged in the transaction are at an arm's length, the terms of the transaction are regulated by a "closed" contract, and the "invisible hand" of the market optimizes the use of resources and minimizes opportunistic behaviour through the price mechanism; this arrangement is appropriate for simple, standardized services/products which have little uncertainty and require minimal

[*] *To appear on the International Journal of CSCW: special issue on CSCW and the Web*"

trust among the actors[1]. At the opposite end of the spectrum there is the team arrangement, which is appropriate for very complex products/services with high uncertainty. In this case, there must be maximal trust and willingness to cooperate among the involved social actors, as they might need to cope with surprises that can emerge during the transaction. In fact, in the absence of a legal contract, the optimal use of resources is ensured by the "invisible hand-shake" among team members, who rely on social mechanisms such as leadership and comradeship to ensure successful coordination and completion of the transaction. In the middle of the spectrum there is the hierarchical arrangement, where the "visible hand" of management coordinates people working under an "open" employment contract. Besides these three main types of organizational arrangements, there are other intermediate forms, such as networks of alliances among companies and virtual enterprises.

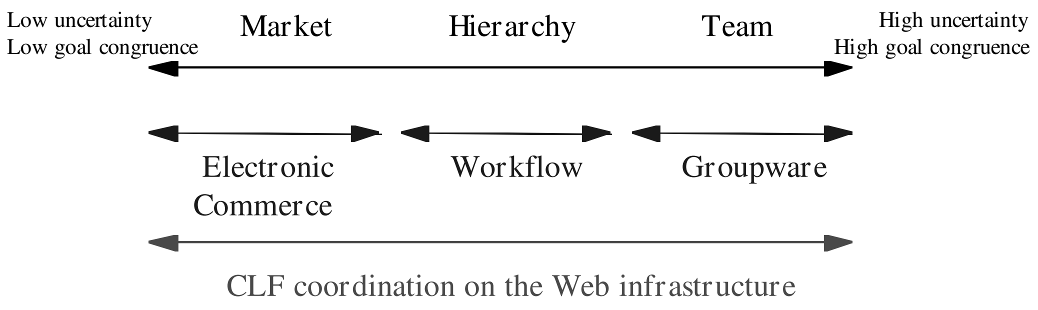


Figure 1: Technology to support market, hierarchical and team arrangements.

Information technology can be viewed as *coordination technology* that changes the structure of transaction costs (Ciborra, 1993), thus allowing organizations to freely shift along the spectrum, picking up the optimal type of organizational arrangement for a given product/service. For example, using IT to support supplier-manufacturer relations, some companies can find that it is more convenient to outsource part of the business (move from hierarchy to market); or, by using groupware technology to support distributed teams, companies can create inter-functional teams for tighter integration across departments (from hierarchy to teams).
The coordination technology available up to now has, however, been specialized to support only a specific organizational arrangement: groupware to support teams, workflow to support the coordination of hierarchical processes, electronic commerce and EDI to support market transactions. Thus, coordination technology is often the enabler that makes a new organizational arrangement possible, but then becomes a barrier to further changes.
By providing an homogeneous medium and computational framework to support communication and coordination over a wide range of organizational arrangements, the World Wide Web and Inter/Intranet infrastructure have the

[1] On the other hand, societies where trust extends to market relationships can have a competitive edge with respect to "trustless" societies. See (Fukuyama, 1996).

potential to reverse this state of affairs. In fact, this homogeneous platform can be used to support business transactions along the entire spectrum, from electronic commerce on the Internet to workflow, knowledge sharing and community building on Intranets. Therefore, the Web enables enterprises to adapt rapidly to changing market conditions by shifting the borders of the enterprise (insourcing and outsourcing), and by adopting tighter or looser coordination mechanisms for internal operations (functional departments, processes, and teams). Furthermore, the Web offers the opportunity to develop new coordination mechanisms to improve customer/supplier relations and internal operations. For example, a building society can offer customers the possibility to apply for a mortgage on the Internet; if the internal mortgage process is managed through the corporate Intranet, the building society can use the same platform to coordinate the various internal departments that need to be involved as well as external third parties (e.g. solicitors), and give customers the possibility to monitor the progress of the mortgage application. Distributed print on demand factories are another example: digital print technologies and the Web enable the shift from "print-and-distribute-on-paper" to "distribute-electronically-and-print": a print job may now be partitioned according to the location where the paper copies are needed (e.g., 10 copies in London, 15 copies in Paris, 5 copies in Milano, etc.) and delegated to various sites such as networked print shops, sub-contractors, or customer's print and computing facilities. These examples show the benefit of using the same platform both within and outside the enterprise: the borders of the company and of functional departments become much easier to shift and blur, by inventing new organizational arrangements to coordinate customers, internal units and suppliers.

2. WWW and the Inter/Intranet: the technological challenge

Another way to understand the organizational impact of the Web is by looking at the practical difficulty of widely distributed user-oriented applications. This is a well-known problem that can be analyzed in terms of the following three sub-problems (Borenstein, 1992):

- The problem of remote installation;
- The problem of user buy-in;
- The problem of heterogeneous user environments.

The wide acceptance of the Web infrastructure has made these issues disappear: a Web application does not require remote installation, provides an interface the user is already familiar with, and is capable of interconnecting users equipped with heterogeneous platforms. This change makes feasible applications that span the whole spectrum of organizational arrangements, from market to teams.

Unfortunately, this is only part of the answer — more is needed, to make the potential of a more flexible and adaptive organizational life into a reality. In fact, the software constructs so far available for application development on the Web belong to the simplest kind of client-server computing. This is not enough for the development of complex applications that require the coordination of multiple components. In particular, the following *coordination* problems need to be tackled:

- *Multi-server, distributed coordination.* The Web is based on a client-server architecture designed for multiple, distributed clients that access a single server. By contrast, the application domains described above involve the coordination of multiple servers. For example, a typical electronic commerce application may require that a client performing a travel reservation coordinates several independent transactions on different servers such as reserving a flight, reserving a car, reserving a hotel room and checking the credit card. Furthermore, the very nature of the Web requires that coordinators communicate with resources distributed over a wide-area network — hence the need to break-down a single logical coordinator into multiple, distributed physical coordinators running at different sites and interacting with local resources. For example, a workflow system supporting a customer-supplier process involves coordinating two workflow engines, one on the customer's premises and one on the supplier's, both responsible for their own local sub-processes. In such an environment, the following technical problems must be addressed: global vs. local consistency, redundant coordinators and resources, tolerance to failures of communication links and computing resources.

- *Flexible, adaptive coordination.* The Web provides an homogeneous platform to support market, hierarchical and team arrangements. It is, however, up to the developer to create applications that are flexible enough to address the different coordination requirements of each arrangement, from highly structured and standardised market transactions to unstructured, document- and knowledge-intensive team relations — hence the need for basic distributed coordination primitives that can be arranged to address different user and system needs.

3. Meeting the challenge: coordination middleware

In the development of WebFlow, we have addressed these needs by adopting the Coordination Language Facility, CLF for short, (Andreoli *et al.,* 1996; Andreoli *et al.,* 1994), a middleware environment for distributed coordination. The CLF provides a basic set of library tools for building *coordinators* and *resource managers*, namely software agents for, respectively, the coordination of complex tasks and the management of resources in distributed environments. On top of the generic modules provided by the CLF we have then developed further modules, such as activity servers and task list managers for distributed workflow, which are specific to the WebFlow environment.

We shall characterize in detail the relationship between WebFlow and the underlying CLF middleware in the course of Section 4. In this section we wish to give a flavour of two key features of the WebFlow environment that strictly rely on the use of the CLF as a supporting infrastructure. Specifically, the WebFlow architecture addresses the issue of distributed, multi-server coordination as two levels, server and client (Figure 2):

On the server side: resource managers are used to extend the standard functionalities of Web servers. Resources represent capabilities of the Web server that can be coordinated; for example, in a document workspace a resource can be a document, or in a workflow system a resource can be an activity. A WebFlow server handles the messages of a standard Web browser, as well as CLF messages encapsulated on the HTTP transport protocol that allow a special client (the coordinator) to insert, inquiry, reserve and consume resources. This approach is similar to the one used in message oriented middleware (MOM) systems, such as IBM MQSeries and Dec MessageQ, that coordinate distributed servers through persistent message queues where messages can be consumed, inserted or browsed (Alonso *et al.,* 1995).

On the client side: the behaviour of coordinators — the clients that implement inter-server coordination — can be itself fine-tuned to meet specific application requirements. The desired coordination pattern is expressed with a rule-based script. Each rule of the script is executed as follows: the coordinator first makes server-by-server *inquiries* for the resources that satisfy given criteria; if they are found, the coordinator tries to *reserve* them; if all reservations succeed, the coordinator commits and transactionally *consumes* all the resources; finally it notifies, if appropriate, other servers that new resources should be *inserted*, as a post-condition of the performed transaction. This simple transactional protocol guarantees that each resource is consumed by one and only one coordinator, even if multiple or redundant coordinators are competing for the same resource. A key property of coordinators is that they are *reflective*, i.e. each rule of the script is a resource that can be inserted or removed by other coordinators in order to change the coordination pattern dynamically[2]. Tailorable, transactional coordinators is what makes the CLF and WebFlow into middleware providing effective coordination capabilities, and not just managing inter-server communication as is the case with simple MOM-based solutions.

---

[2] Borghoff *et al*. (1996a) illustrate the use of the reflective features of coordinators to handle dynamic changes of business policies in workflow management (Ellis *et al.,* 1995).

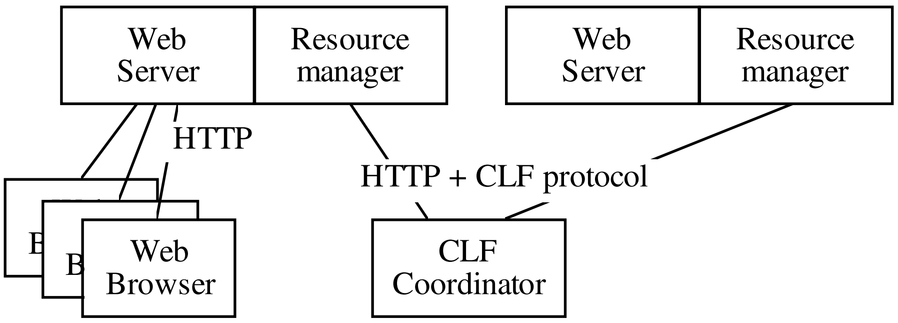


Figure 2: WebFlow supports distributed, multi-server coordination by extending standard Web servers with "resource managers" and by providing coordinator clients.

To get an idea of what this means from the point of view of application development, consider the following "documents-on-demand" application: consistently check out a compound document doc(A1, B1) made up of two chapters doc(A1, version) and doc(B1, version) stored on two different document workspaces (Figure 3). With the CLF/WebFlow approach, we can extend both workspace servers with resource managers that map (versions of) documents into resources. When the user connects to Workspace Server C with a Web browser and requests to check out document doc(A1,B1), two coordinators are created that perform inquiries, respectively, for doc(A1,X) on server A and for doc(B1,Y) on server B. For instance, the inquiry on A may return two resources corresponding to two different versions of A1, while the inquiry on B may return one single resource. The coordinator then picks up resources as appropriate, for example those corresponding to the last version of each chapter, and tries to reserve them. The coordinator may be competing with other coordinators or users who are also trying to check out the same documents, so the reservation can fail. If it is successful on *both* documents, both resources will be consumed transactionally, thus preventing other users or coordinators to take them. doc(A1,B1) is then inserted into workspace server C and the user is notified of its availability.

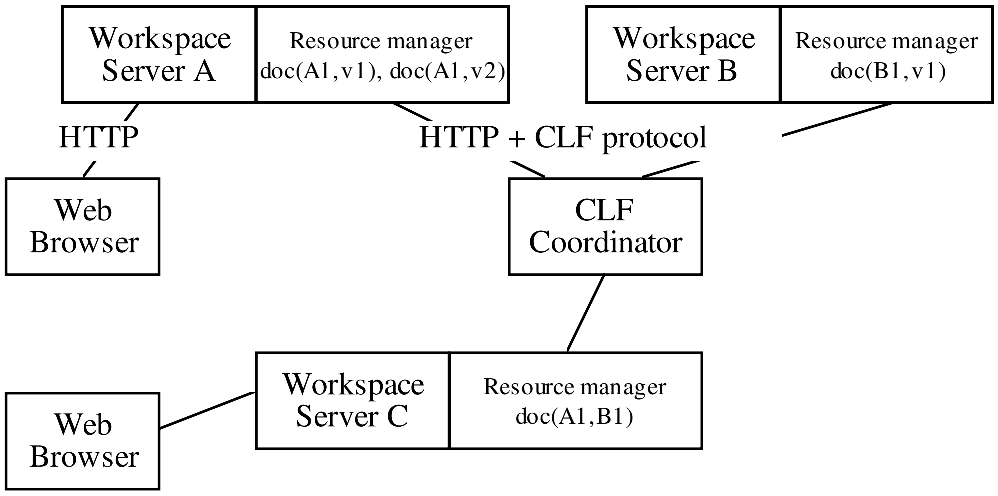


Figure 3: Handling consistency among distributed workspaces.

4. Distributed workflow in the WebFlow environment

For the rest of the paper we focus on the "distributed workflow" component of the WebFlow environment. By "distributed workflow" we mean workflow management for the distributed enterprise, where multiple sites and a mobile workforce interact through the corporate Intranet.

We have seen that, in our spectrum of organizational arrangements (see Section 1), workflow occupies the middle position, corresponding to the situation of hierarchically managed internal processes. Even so, local management of work in the different sites of distributed organizations defines a major paradigm shift with respect to the usual way of supporting work processes, which has in the past been based on the assumption that there should be a single repository for information in the enterprise. Distributed workflow requires in fact distribution of computing and information storage throughout the organization. The resulting form of process support should *(i)* enable operational autonomy in a process-centric running of business operations, *(ii)* eliminate the performance bottlenecks and global server failures that affect workflow architectures where process execution is managed through single servers, *(iii)* allow free interplay of process-based "structured" work with collaborative "un-structured" work by providing a seamless integration with communication and document sharing services. Particular attention needs also to be paid to providing adequate capabilities for software and model engineering that effectively capture the distributed aspects of the application development environment.

In the WebFlow environment we address these issues through three different phases of workflow development: design, deployment and enactment. During the design phase, a process graphical editor allows users to design cooperatively the processes, thus fulfilling the requirements of distributed design. A translation of the graphical maps produces the set of CLF rules which are needed to coordinate the distributed execution of the process. The produced set of rules expresses the control/data flow among activities. Once the process design phase is over, a deployment phase takes place: all activity definitions and generated rules are transferred and installed at their appropriate sites. Each site may in turn have its own policy about specialized control and about access to third-party software components such as groupware and user applications. This aspect is supported by the object-oriented features and scripting capabilities of the CLF, that make it possible to quickly reconfigure the way the software handles business processes and workflows — without changing the implementation of specific components. Then, at enactment time, each place manages its own part of the process and interacts with the other places to receive and send, respectively, input and output work items. In this way, by avoiding the need of communicating with the server during execution, the system is both more efficient and resilient to failures.

Figure 4 shows how the three phases of workflow development corresponding to design (a), deployment (b) and enactment (c) are linked together in WebFlow.

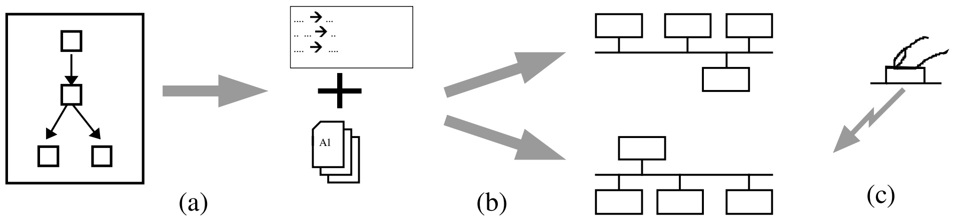


Figure 4: Workflow development phases.

4.1 ***Workflow design***

Our implementation of the process editor is directly influenced by our view of process modeling, which to a large extent blurs the distinction between repetitive processes and ad-hoc processes that has been made popular by workflow trade classifications. According to this taxonomy, typical examples of repetitive processes are insurance claims processing, loans origination and accounts payable. Typical examples of ad-hoc processes are the generation of a strategic plan or the development of an advertising campaign. This distinction is however increasingly showing its limitations, as it is becoming clear that most organizations need integration of both kinds of workflow. In fact, for most organizations, processes that are completely repetitive or completely ad-hoc do not exist. Rather, both aspects coexist in a process and must be managed in the final workflow solution. For instance, insurance companies highest risk/highest revenue procedures are those related to the reimbursement of large damages; these procedures are highly structured and formalized and, as a matter of fact, are legally auditable. Specific instances may require, however, triggering ad-hoc sub-processes to cope with problems requiring the help and the knowledge of experts.

By contrast, rather than drawing the distinction in terms of different types of processes, we draw it in terms of different types of process representations, and we work under the assumption that "plans (processes) are resources for action", as come out from ethnographic studies on the nature of work (Suchman, 1987). According to this view, the distinction between repetitive, procedural processes and ad-hoc, exceptional ones, is misleading. On the other hand, there can be different representations of the same process, fulfilling different needs. In fact, process representations are viewed as artifacts used by the actors of the process to accomplish their work, enabling them to dynamically refine the processes and thus create knowledge by incremental design. But they are also (Suchman, 1987; Bowers *et al.,* 1995) means used by organizations to monitor and control the progress of work — hence the need of process representations capable to conciliate this intrinsic ambiguity[3].

[3] For instance, (MacLean and Marqvardsen, 1996) proposes to distinguish between *normative*, *routinary* and *ad-hoc* descriptions of the mortgage application process in a multi-site financial organization. The normative description is the official process map used by the organization to define and monitor the process. The routinary description refines the normative one by extending it with activities developed in the course of time through everyday practice, e.g. screening customer information and answering customer

Our cooperative process editor has been designed on the basis of these principles to enable distributed organizations to flexibly blend centralized control with local autonomy, dynamically change business processes, broaden decision-making authority and extend business processes to casual users, suppliers, partners and customers. The aspects of the implementation that capture the requirements above can be summarized as follows:

- Processes are defined in the usual way as flows of activities, assignment of roles, temporal and document scoping, to provide general process maps that allow organisations to monitor the work and enable workers to put their activities in context.

- Process design is distributed across the sites of the distributed enterprise, making sure that work gets described where it has to be accomplished. In particular, processes can be specialised via the definition of local sub-processes.

- Process descriptions may evolve through time, by allowing users to extend and modify them.

- Java applets can migrate into the environment of "casual" users, to provide them with the functionalities to create and modify the process.

Co-editing is achieved through the use of a Web-based collaborative environment, the Basic Support for Cooperative Work (BSCW) shared workspace system (Bentley *et al.,* 1997a; Bentley *et al.,* 1997b)**.**

4.1.1 **Process and activity models**

The main objects handled by the process editor are processes, sub-processes and activities. An activity object is associated with the specification of all or of a subset of the following entities:

- an **Agent**. An agent can be a role, an actor or a group; in the latter case the agent can be specified as anyone in the group or as the group in its totality.
- a **Site** among the registered ones.
- a set of **Documents** (with associated meta-information concerning their creation, modifications etc.).
- an estimated **Duration** and a **Deadline**.

---

questions. Finally, the ad-hoc representation captures less frequent events that can still be related to the normative and routinary process descriptions. MacLean points out how different teams implement different routinary processes while still being able to locate their work in the context of the same normative description, as "*there can be considerable variation between different units which have the same role in the process*" and the normative description "*provides a framework which allows different ways of working to emerge in different places which have the same role in terms of responsibilities for business processes*".

An activity can be a sub-process in itself, and different activities can relate to each other, either by running in parallel or by being dependent one another. The semantics of dependencies among activities is both causal and temporal: if activity **Q** is dependent on activity **P**, this means that **Q** needs input produced by **P** and that **Q** cannot start until **P** is totally finished.
Each individual activity is modeled as having three basic states:

- inactive: the activity cannot be executed because the previous dependencies have not been satisfied yet;

- active: the previous dependencies have been satisfied so the activity can be processed;

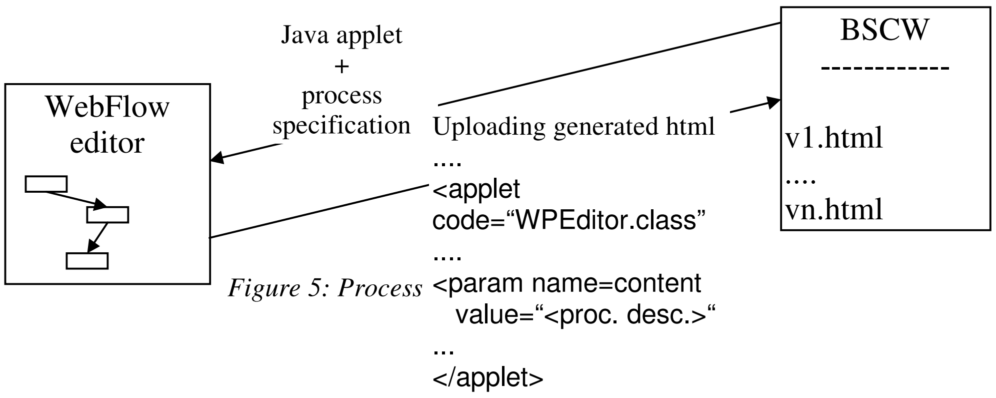
- done: the activity has produced the desired output and its dependent activities can be triggered.

This way of modeling activity dependencies and activity states is the usual one adopted in workflow management systems, and has been often criticized for its rigidity and lack of tailorability. More recently, richer modeling frameworks have been proposed (Dourish *et al.* 1996; Glance *et al.* 1996) and we plan to integrate them in future versions of the system.

4.1.2 **Process co-editing**

Co-editing of documents has been widely studied in the CSCW literature, with case studies that highlight the different ways of doing work (e.g. see Posner and Baeker, 1992), and with system architectures that focus on consistency and concurrency aspects (e.g. see Pacull *et al.*, 1994 and the paragraph about concurrency in Ellis *et al.*, 1991). Nevertheless, little work exists that is specific to the co-editing of process maps, though this aspect is crucial in distributing a process across different sites, since the knowledge about the corresponding sub-processes resides on the sites themselves. Therefore, designing distributed processes requires the possibility to collaborate around them, to share them as resources, to discuss their definition, and to be aware of “remote” changes made by others.
Among several collaborative environments for the Web that are starting to become available, we have chosen the BSCW system, developed at GMD (Bentley *et al.,* 1997a; Bentley *et al.,* 1997b). This system allows sharing, across workspaces, documents that can be discussed and versioned by workspace members. Therefore it is used within WebFlow to support the setting up of dynamic groups of process designers, which define and version the process descriptions on the basis of the knowledge they have of the group or organization they belong to. Moreover, we make use of the Java remote programming language to make available the process editor as a Java applet. This feature is particularly useful for supporting “casual” users who may not have the process editor on their own client machines and is implemented by displaying process descriptions as HTML documents in the BSCW system, from which the process editor applet can be started. The editor application is embedded in each process

map by means of a Java applet, and the applet holds as parameters the actual location of the graphical and logical data describing the process (Figure 5)[4].



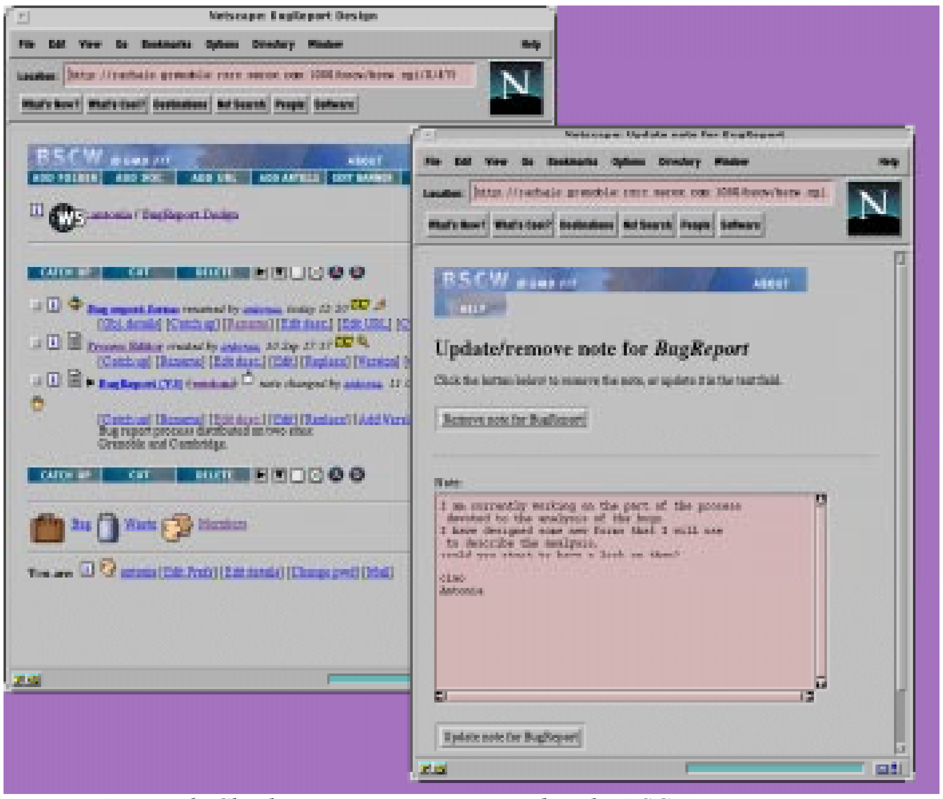
```
....
<applet
code="WPEditor.class"
....
<param name=content
  value="<proc. desc.>"
...
</applet>
```

Figure 5: Process

Furthermore, Java allows an easy integration of the WebFlow process editor and the BSCW services (see Figure 6 and Figure 7), thus leveraging the cooperative services provided by BSCW for versioning and soft check-out (Bentley *et al.*, 1997a):

- from a workspace, it is possible to create a new process description from the home page of the graphical editor, which is part of the WebFlow environment;
- for every process, a shared workspace (created by a group of process designers) holds the different versions of its process map;
- new process versions are uploaded as new BSCW versions of the document;
- when concurrent editing occurs, the involved users are warned by a BSCW note and it is left to them how to solve possible conflicts.

[4] In the actual implementation a proprietary format for describing the processes is used; this will be substituted by a higher level formalism such as the GPSG grammars (Glance *et al.*, 1996).


Update/remove note for *BugReport*

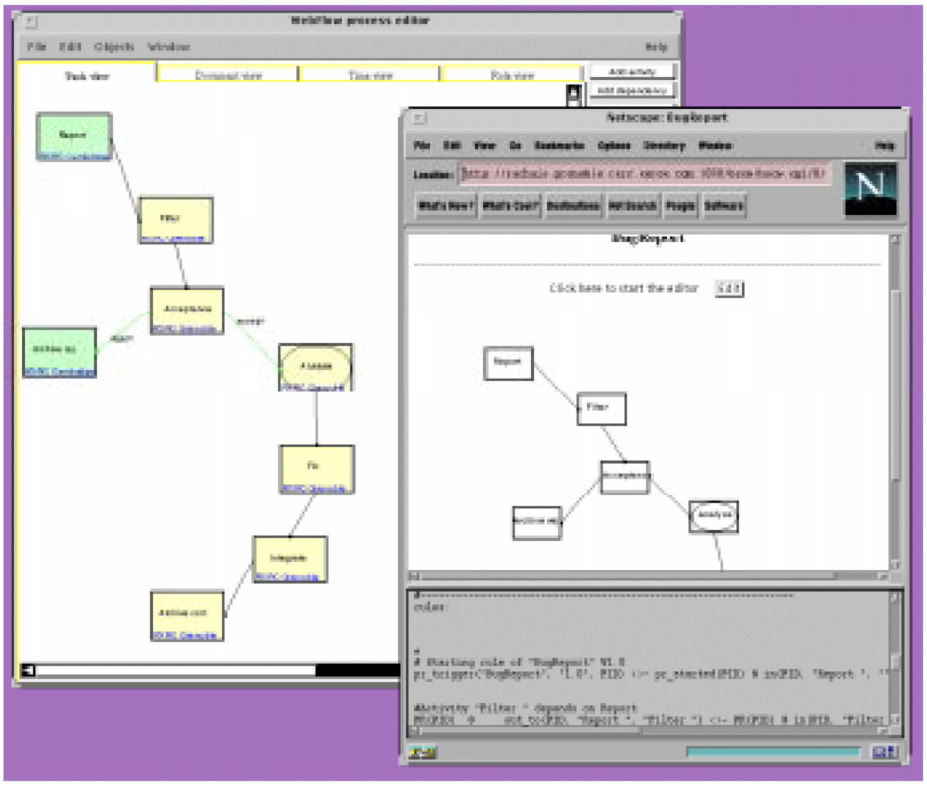
Click the button below to remove the note, or update it in the text field.

Note:

```
I am currently working on the part of the process
 devoted to the analysis of the bugs.
I have designed some new forms that I will use
 to describe the analysis.
would you start to have a look at them?

ciao
Antonio
```

Figure 6: Checking out a process stored in the BSCW environment.



```
#-----------------------------------------------------------
rules:

#
# Starting rule of "BugReport" V1.0
pr_trigger("BugReport", "1.0", PID <- pr_started(PID & inPID, "Report ", ""

#Activity "Filter " depends on Report
```

Figure 7: Editing a process stored in the BSCW environment.

4.2 *Workflow deployment and document management services*

In distributed workflow, deployment corresponds to assigning the business/process rules to the appropriate sites. To a large extent, this is achieved within WebFlow as a direct consequence of the fact that process design has been implemented in such a way that each site is made responsible for the definition of its own portion of the process.

Another relevant aspect of deployment is the interaction between the workflow management system and other applications like databases, legacy systems, collaborative tools and, in particular, Document Management Systems (DMS), since documents are the primary objects manipulated by workflows. Indeed, as part of the distributed workflow architecture, we need a server to handle the documents created and/or modified in the course of a process. This server is an object that can talk, like the others servers in our workflow configuration, both the CLF and the HTTP protocol. In the actual implementation an *ad hoc* DMS wrapper has been developed, but the objective is to use a general (in the sense of not depending on any specific DMS) document-oriented API for interfacing the CLF with document managers; to achieve this objective we are currently looking at the API of the Document Management Alliance (DMA) standard (DMA Technical Committee, 1995).

The DMS we use for storing the documents manipulated *by* the process is BSCW again, just as in the previous section we illustrated the use of BSCW for storing documents *about* the process. The integration of the process engine with a collaborative DMS such as BCSW brings in for free collaborative capabilities such as sharing documents and holding contextualized discussions.

Another relevant aspect of document management closely interwoven with workflow deployment is the need to support distribution of documents. In this sense, distribution of documents is simply the movement of the document source to the location where work gets done. The challenge is to provide seamless access to information regardless of document location. Distribution of documents should be handled at an architectural level and made transparent to the end user, achieving efficient use of network services and optimization of local data access.

This aspect is not yet supported in the current implementation of WebFlow, but the idea is to provide it in future versions by implementing replication capabilities for BSCW. Thus, individual sites can subscribe to a subset of the central document base and access that information locally. With local copies of documents for use, users can do their work without having to log into the central document repository that may be remotely located. Furthermore, local users are likely to receive the results of their queries faster, since they are accessing local (replicated) rather than remote documents. Conversely, this replication service should be complemented by a simple mechanism to consolidate distributed information in a single logical place. For this purpose, one could use “broker agents” (Borghoff, 1996a; Borghoff, 1996b) for information gathering in distributed environments.

4.3 ***Enactment.***

The need to decentralize information sources in distributed workflow is even more crucial for workflow execution than for document access. In fact, distributed workflow cannot be adequately enacted through the standard client-server architectures where workflow information resides on one centralized database, for two main reasons:

- in a distributed enterprise where work is performed at multiple sites, most users will need to access the central server remotely, thus creating the potential for huge bottlenecks;
- server failure will globally affect all currently active operations — an unacceptable price to pay for business processes that span wide geographical areas.

With WebFlow, we have aimed to meet these requirements by implementing a workflow enactment architecture on the WWW with the following two main features:

i) The workflow state is stored on one or more specialized WWW servers, called *process servers*. WebFlow (CLF) coordinators enact the workflow by engaging negotiations among distributed process servers and other network services such as document repositories, user directories, databases, etc. Coordinators communicate with process servers using the HTTP protocol; therefore coordinators look like WWW browsers from the process server's point of view (see Figure 8). This differs from the standard approach of adding workflow functionality to the Web by connecting a workflow engine to a WWW server through the CGI interface (see Figure 9). We believe that our architecture, besides making the implementation much easier, can scale up to support truly distributed business processes that span multiple process servers in different departments and organizations. Furthermore, the coordinator-as-client architecture can better support electronic commerce applications.

ii) Users can easily interact with the process through a standard WWW browser connected to a process server, without the need to install a specialized workflow client. Thus, WebFlow supports the enactment of processes with multiple levels of users: those with the specialized workflow client application; those with a WWW browser augmented with computation features (e.g., a Java interpreter); and those with a standard WWW browser. Similarly to the approach we have taken for process design, we exploit Java applets at process execution time both to make users aware of what is going on in the process (e.g., by notifying users of new activities or by devising forms of synchronous communication to connect users working on different parts of the process), and to provide more advanced interface functionalities with respect to HTML pages.

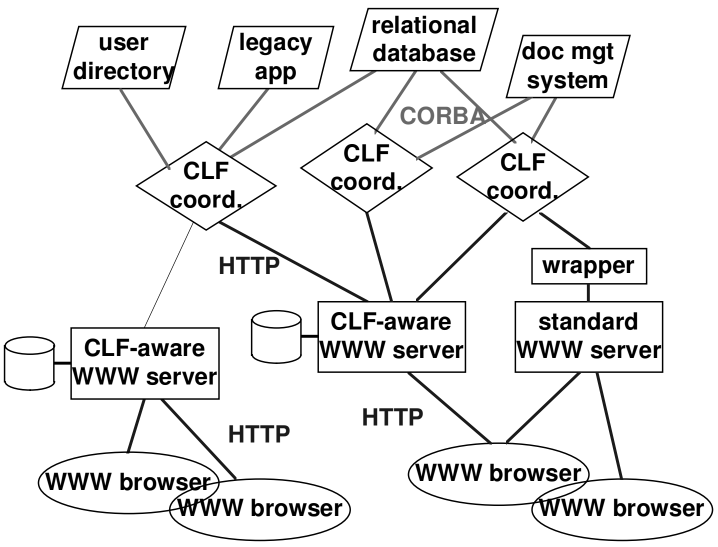


Figure 8: WebFlow architecture: multiple work servers are coordinated by multiple CLF coordinators, that can interact with other software objects using multiple protocols (CORBA, OLE/COM, etc.).

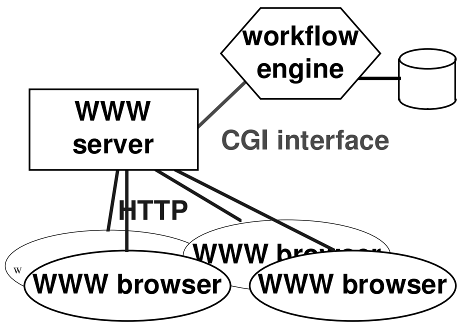


Figure 9: Traditional architecture of a workflow engine connected to a Web server through the CGI interface.

4.3.1 **CLF: a quick tour**

We now go into the details of the CLF infrastructure that provides the basis for the WebFlow functionalities. The CLF is based on an extended ***object model*** that assumes that all objects can be viewed as ***resource managers***, accepting two basic types of operations: removal and insertion of specified resources. Resources are not directly visible from the outside world but can be accessed through their properties, defined in the interface of the object.
The underlying architecture is client/server where the client tries to insert/remove resources from the server. The following protocol defines the allowed client/server interactions:

Inquiry: the client inquires whether the server holds (or can produce) a resource satisfying a given property specified in the server's interface. The server returns a *stream* of actions that it could perform in order to produce such a resource. The returned stream may be empty (no action can produce a resource satisfying the given property, neither now nor in the future) or infinite (a new action is added to the stream each time the server changes its state and a new resource satisfying the given property becomes available). This phase of the interaction is therefore *deferred synchronous*.

Reservation: the client asks the server to reserve one of the actions returned during the *Inquiry* phase. If an action is successfully reserved, the server commits to perform it on request (i.e. to remove the corresponding resource). Failure may occur when an action returned in the *Inquiry* phase is no longer available at the time of the *Reservation*. This phase of the interaction returns one result (success or failure) and is therefore *synchronous*.

Confirmation/Cancellation: the client may either confirm or cancel an action it has successfully reserved. If the reservation is confirmed, the action must be executed, leading to the deletion of the corresponding resource; if canceled, the action may become available again to other reservation requests. No result is expected in this phase of the interaction, which cannot fail: it is purely *asynchronous*.

Insert: the client requests to insert into the server some resources satisfying a given property. This is also purely *asynchronous*.

There are other operations in the protocol, which we omit here for the sake of simplicity, and in particular *Check* and *Kill* which allow garbage collection, in a server, of *Inquiries* that are no longer of interest for a client. Notice that the *Inquiry* operation starts a long-lived process on the server, in charge of warning the client each time a resource satisfying the given property becomes available.

The *Inquiry* operations of the protocol provide the basis for a negotiation phase where a client asks a server to make its offerings known. The Reserve/Confirm/Cancel operations of the protocol provide the basis for elementary transactions based on a two-phase commit protocol: a set of actions can be performed atomically by reserving each of them separately, then either confirm them all if all reservations have succeeded or cancel the successful ones if some reservation has failed.

Client/server interactions are specified in the CLF through a rule-based scripting language, and rule sets define ***coordinator agents***, that implement clients' behavior. Both sides of a CLF rule take the form of lists of symbolic *tokens* which represent different *properties* of the *resources* held by the CLF objects. Basically, a token identifies a "resource bank" in a given server. The tokens on the left-hand side request withdrawal of resources from the corresponding banks, while the tokens on the right-hand side deposit new resources, if the withdrawals on the left-hand side have been successfully performed. *The transactional unit is* *the rule*, thus it is guaranteed that the removals and insertions are performed atomically.

Formally, the left- and right-hand sides of a rule are separated by the symbol <>- , and are composed of lists of tokens separated by the symbol @. The abstract syntax for rules is as follows:

```
Rule           = TokenList <>- TokenList

TokenList      = Token
               | Token @ TokenList

Token          = TokenName ( ParameterList )

ParameterList  = ParameterName
               | ParameterName, ParameterList
```

Thus, a rule of the form **`p(X,Y) @ q(Y,Z) <>- r(X,Z)`** will try to (i) find a resource satisfying the property **`p(X,Y)`** and a resource satisfying the property **`q(Y,Z)`** for consistent values of **`X,Y,Z`**, then (ii) extract these two resources atomically, and finally (iii) insert a resource satisfying **`r(X,Z)`**. We assume here that the token names **`p,q,r`** refer, through a given name service, to properties declared in the interface (i.e. in a bank) of some resource manager.

Apart from rules, coordinators also handle signature resources, to specify input/output dependencies between different *Inquiry* operations. This is simply achieved by letting a signature for a given token name specify whether a given parameter needs to be already instantiated for an *Inquiry* on this parameter to be authorized. The abstract syntax for signatures is:

```
Signature      = Token : ParamListNil -> ParamListNil

ParamListNil   = ParameterList
               | nil
```

For example, a signature like **`q(X,Y):x->y`** may say that resources satisfying the property **`q(X,Y)`** for some values of **`X,Y`**, can be inquired through that token only if **`X`** has already been defined; if this is the case, every answer to *Inquiry* returns then a possible value for **`Y`**.

In the definition of a coordinator the developer may find it useful to perform simple computations or tests with resources, for instance creating a new resource from the concatenation of two existing resources, or testing if a resource is greater than a given value. For this reason, the CLF scripting language supports the option to associate in-line code with a token.

With the CLF, the development of distributed applications is supported in a number of new ways. First, coordinators are *idempotent*: several coordinators defined by the same set of rules have the same behavior as if only one were running. Indeed, due to the transactional properties of resource managers, a resource may be consumed only once, so only one of the competing coordinators will be able to grab it. Thus, coordinators can be safely replicated to obtain reliable behavior. Second, coordinators are *compositional*: combining together different coordinators produces a coordinator whose behavior is equal to the sum of the combined coordinators' behaviors. This means that a complex coordinator can be split into smaller coordinators assigned to different sites and locations, and that coordinator modules can be independently developed and then combined in a larger application. Finally, the transactional properties of coordinators make sure that resources can be inspected, inserted, and removed without creating any inconsistency.

4.3.2 **Workflow servers and browser/coordinator clients**

The architecture of our system leverages the *object* and *rule* constructs of the CLF. The resources of our system are of the following types: process template, process, activity definition, activity, users, documents (see Figure 12). There are six types of CLF objects, that together implement the super-type of *CLF-aware WWW-servers*:

1. The **process template servers** manage the process editor and the other design resources.

2. The **process servers** manage the state of the various processes being enacted.

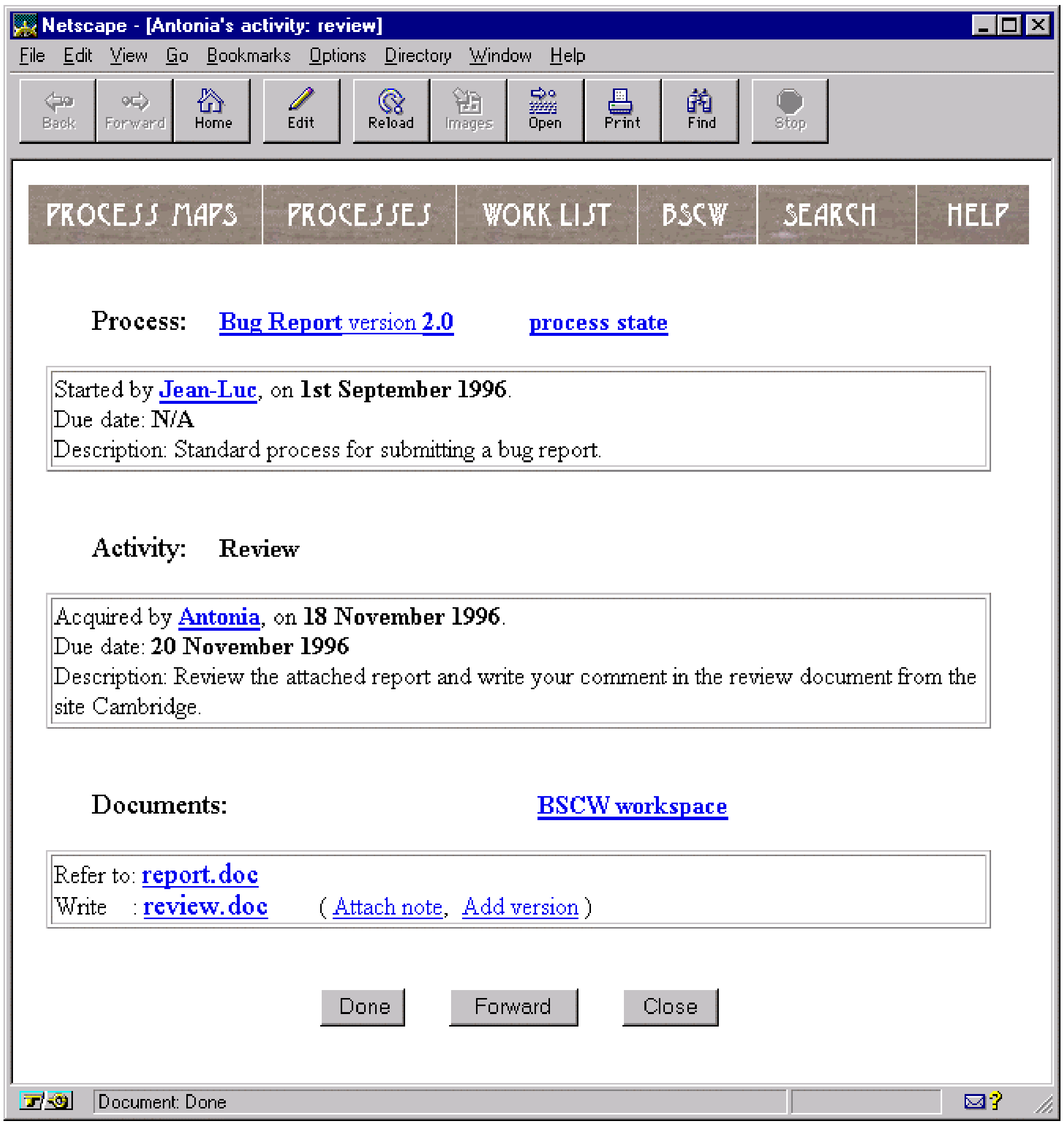
3. The **activity servers** manage states and properties of the tasks to be performed.

4. **User directories** are obtained by wrapping up pre-existing user directories.

5. The **document servers** are obtained by wrapping up off-the-shell document management systems — BSCW is the single DMS supported in the current implementation of WebFlow.

6. **Task list managers** (**TLM** for short) do not really manage resources but deal with the execution of activities.

All these servers accept CLF coordinators or Web browsers as clients. For instance, CLF coordinators insert activities into users' TLMs, and users can view and access the TLMs through browsers. Figure 10 shows a work item selected by the user Antonia from her task list displayed with a description of the activity and pointers to the relevant documents. These documents are stored in a BSCW workspace. Antonia has been automatically invited to join this workspace and she can in turn invite other users if needed.


Netscape - [Antonia's activity: review]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Edit Reload Images Open Print Find Stop
PROCESS MAPS PROCESSES WORK LIST BSCW SEARCH HELP

Process: Bug Report version 2.0 process state

Started by Jean-Luc, on 1st September 1996.
Due date: N/A
Description: Standard process for submitting a bug report.

Activity: Review

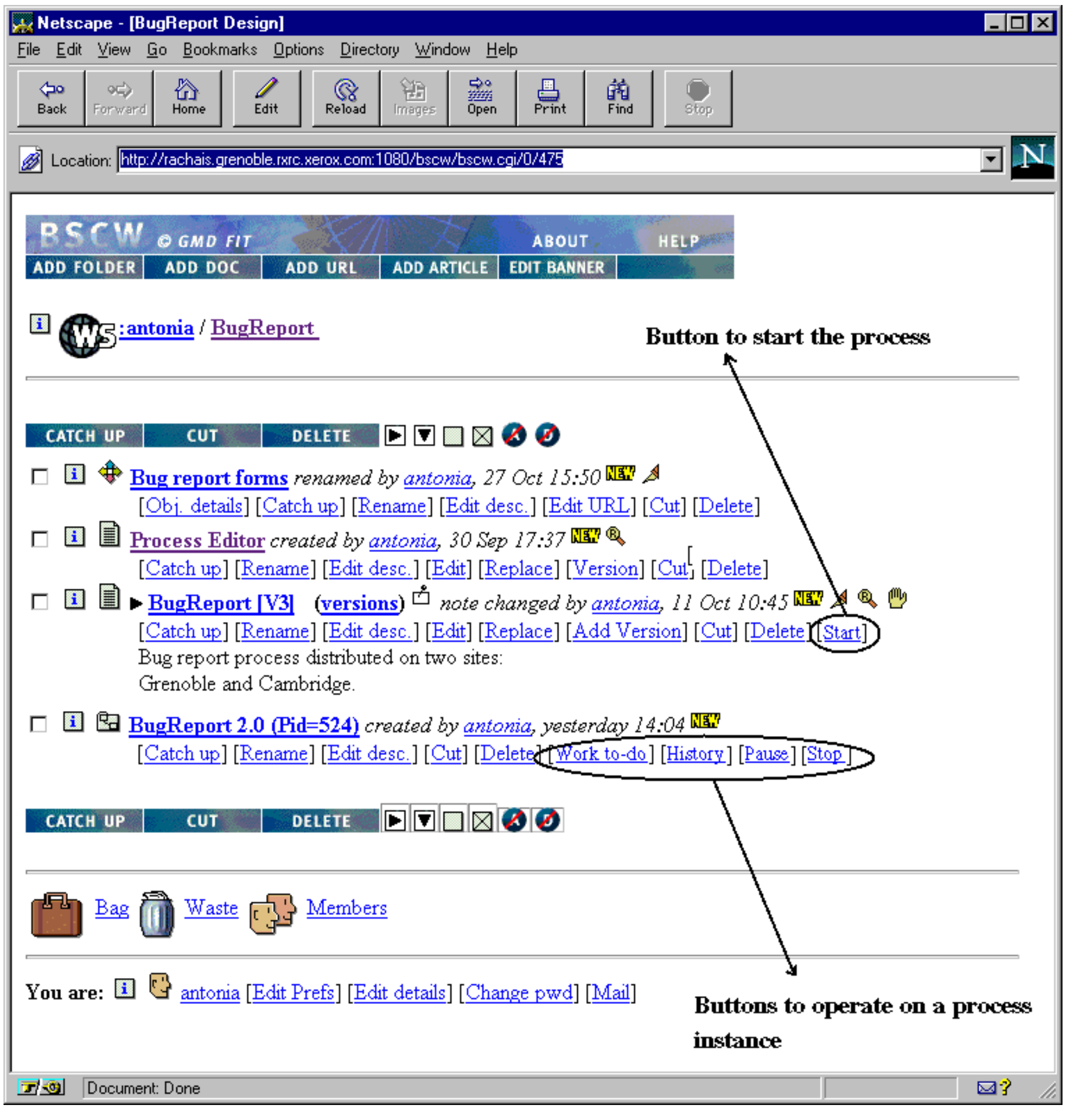
Acquired by Antonia, on 18 November 1996.
Due date: 20 November 1996
Description: Review the attached report and write your comment in the review document from the site Cambridge.

Documents: BSCW workspace

Refer to: report.doc
Write : review.doc (Attach note, Add version)

Figure 10: WebFlow work item.

In a similar way, a process server displays the state of the executing process instances to the relevant users. By complying to the CLF protocol, it also allows a coordinator to act directly on process instances by querying, inserting, and removing resources.
As shown in Figure 11, WebFlow can be called directly from the BSCW environment, by adding the 'start' action for process execution to the BSCW interface to enact a process and by adding process objects with specific actions.


Netscape - [BugReport Design]

File Edit View Go Bookmarks Options Directory Window Help

Location: http://rachais.grenoble.rxrc.xerox.com:1080/bscw/bscw.cgi/0/475

:antonia / BugReport

Bug report forms *renamed by antonia, 27 Oct 15:50*
[Obj. details] [Catch up] [Rename] [Edit desc.] [Edit URL] [Cut] [Delete]

Process Editor *created by antonia, 30 Sep 17:37*
[Catch up] [Rename] [Edit desc.] [Edit] [Replace] [Version] [Cut] [Delete]

BugReport [V3] (versions) *note changed by antonia, 11 Oct 10:45*
[Catch up] [Rename] [Edit desc.] [Edit] [Replace] [Add Version] [Cut] [Delete] [Start]
Bug report process distributed on two sites:
Grenoble and Cambridge.

BugReport 2.0 (Pid=524) *created by antonia, yesterday 14:04*
[Catch up] [Rename] [Edit desc.] [Cut] [Delete] [Work to-do] [History] [Pause] [Stop]

Bag Waste Members

You are: antonia [Edit Prefs] [Edit details] [Change pwd] [Mail]

Document: Done

Figure 11: BSCW/WebFlow integrated environment.

To get the system running, at least one server of each kind should be in existence. Therefore, the simplest configuration is shown in the following figure:

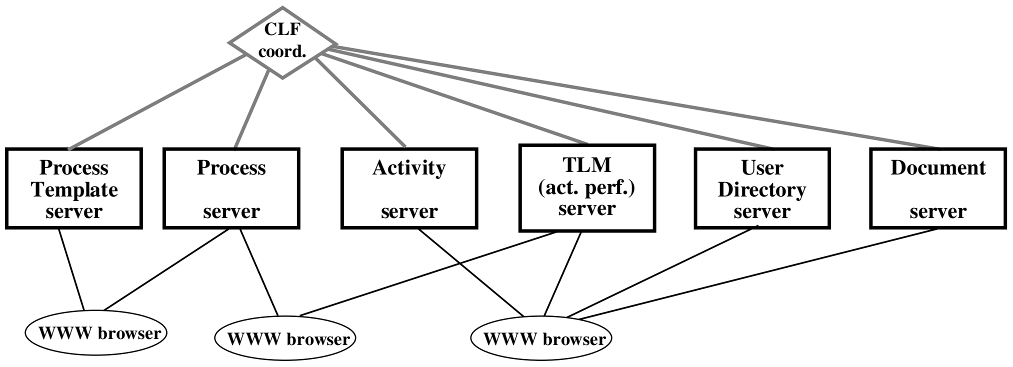


Figure 12: WebFlow servers.

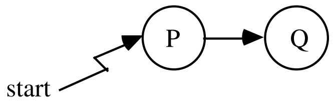
Inter-server coordination is obtained through three kinds of CLF rules, corresponding to three types of coordination:

- ***coordination of the flow:*** pacing the activation of activities as specified in the network of dependencies among activities.
- ***coordination of the work:*** supporting and monitoring the execution of each activity.
- ***coordination of the sites:*** maintaining consistency between sites and offering functions such as process creation, etc.

Specifically, **coordinating the flow** deals with the coordination of transitions which remove resources corresponding to the output availability of some processes and insert resources corresponding to the input availability of other processes. Such transitions can straightforwardly be encoded into CLF rules. As an example, we show below how to model traditional imperative coordination constructs. The rules introduced here are only abstract skeletons, which need to be concretized by appropriately choosing the tokens for each resource.

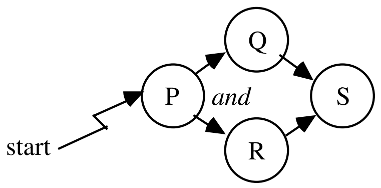
Sequential execution

P-out <>– Q-in



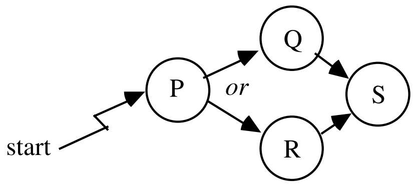
Parallel execution, rendezvous

P-out <>– Q-in @ R-in
Q-out @ R-out <>– S-in



Branching

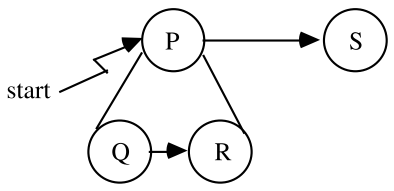
P-out @ case1 <>– Q-in
P-out @ case2 <>– R-in
Q-out <>– S-in
R-out <>– S-in



Here **case1** and **case2** represent resources which are supposed to be mutually exclusive and allow conditional choice between the two branches. Typically, the tokens for **P-out** and **case1/case2** share a common variable (output for the former, input for the latter), and the selection tokens simply perform mutually exclusive tests on this variable (e.g. a test and its negation).

Decomposition

P-in	**<>–**	**Q-in**
Q-out	**<>–**	**R-in**
R-out	**<>–**	**P-out**



The rules pertaining to the support of a process template are made of such constructs by the Process Template server. The **in** and **out** resources are instead managed by the Activity servers.
As a CLF rule defines a transaction, the consistency of the system is always maintained, even if a coordination construct involves several distributed servers. In this case, the best execution for the rule is as close as possible to the activity server. Therefore it is deployed over the (possibly several) closest coordinator(s). This is one of the purposes of the deployment phase.

Coordinating the work consists of finding authorized users available to perform any activity which is *active* and gathering the related information, such as involved documents, textual description, deadline, etc.
For instance, an activity assignment policy may offer *active* activities to users who have the right role or skills and and then assign it to the first one who agrees to perform it. Offering tasks to users is done via a rule of the form:

P-in @ P-propose <>– P-acquired

When a user accepts the tasks, the **P-propose** resource is created and the rule fired, with the effect of inserting a resource **P-acquired** indicating that the task has been assigned to the user. Competition in task acquisition is managed through the transactionality of CLF rules.
Tasks performance is expressed by:

P-acquired @ P-exec <>– P-out

When the user notifies the system that the task is done, a **P-exec** resource is created and the rule fired.
The two rule skeletons above define the interaction between a Task List Manager and an Activity server. By adding more rules of this type a Task List Manager can be connected to several Activity servers. With similar flexibility, we can define multiple policies for task allocation to users.

Coordinating the sites concerns coordinating servers that may be located over different sites**.** For instance we may choose to duplicate information about running processes so to preserve maximal site autonomy. In particular, the status of a process (*started*, *paused*, *ended*) is replicated on every coordinated site because this information is checked very often during enactment while being rarely modified. One case of modification occurs when an administrator wants to freeze the process, i.e. wants to set its status to *paused*. We manage this update by one dedicated CLF rule**:**

S1-P-started @ S2-P-started @ Admin-Pause-Request

<>– S1-P-paused @ S2-P-paused

Cross-site coordination occurs also after the process design phase is finished, because the transfer of generated rules and activity definition is again coordinated by CLF rules, thus exploiting the reflective capabilities of the CLF.

Generally speaking, the initial system configuration can be incrementally extended by adding new servers. These can be easily integrated by defining new CLF scripts. For instance:

- The addition of redundant TLM(s), to let users access their task list through several servers, thus removing the potential bottleneck.
- The addition of new TLMs and user directories: in this way, distinct groups of users have their own server, while still sharing the same activity and process server.
- The addition of document servers: several distinct DMSs participate in the management of the documents needed for the workflow.

In this way, the configuration of the system can be tuned to capture organizational changes and rearrangements. For instance, connecting distinct groups of users can be achieved by adding new TLMs and possibly new user directories. Adding new activity servers will empower groups by providing them with computational resources to manage their own tasks. Further empowerment will result by giving groups their own process servers, thus granting ownership of the management of their part of the process.

5. Related work

Of the many products and projects that relate to distributed workflow, four seem particularly relevant for comparison with our approach: Regatta, Pavone GroupFlow, Exotica/FMQM and the Task Manager.

Regatta

In the Regatta system (Swenson, 1993) , local refinement of the process is possible for every user. This feature introduces flexibility by giving users freedom in the way they carry out personal work, while the system provides them with the interface to attach themselves to the part of the process already defined. This approach fits even better with processes distributed across different organizational sites, because no centralized knowledge could be used to define a decentralized process in all of its parts. Thus, we could say that in our case the Regatta approach has been enlarged in scale, by moving from users and user groups to geographical sites and organizational units.

Pavone GroupFlow

Given the on-going lively debate on whether Notes or the Web will be the collaborative environment of the future (see Ginsburg and Duliba, 1997 for one perspective on this), another interesting comparison is with the GroupFlow product from Pavone, which is built on top of Lotus Notes.
The main aim in the design of GroupFlow has been to put higher level primitives for designing and enacting workflows on top of the Notes functionalities and APIs. The main achievement of this effort is that a collaborative environment for unstructured work such as Notes becomes seamlessly integrated with a process-oriented environment, with graphical design and simulation tools. Moreover, users are given the option to freely switch from structured to unstructured work by leaving workflow, starting a Notes collaborative session and then resume the process. We have followed a similar approach in integrating workflow with the Web and BSCW, but in addition we have provided capabilities for distributing workflow that are missing from Pavone GroupFlow.

Exotica/FMQM
Exotica/FMQM (Alonso *et al.,* 1995) is a distributed workflow prototype (based on IBM's workflow product FlowMark) where distribution is achieved through the use of message-oriented middleware (MOM). The use of MOM in Exotica/FMQM is similar to our use of the CLF for supporting inter-server communication during workflow execution. However, with respect to MOM, the use of a full-blown scripting language for coordination like the CLF offers the advantages of more flexible specification of the interaction protocols and provides capabilities for inter-node migration of activity definitions. Moreover, the CLF provides "light-weight" built-in transactional capabilities (two-phase commit), that in general can be achieved in MOM only through the integration with Transaction Processing (TP) monitors. These capabilities are particularly useful for maintaining system consistency of distributed workflow, and for integrating with legacy systems.

Task Manager
The Task Manager (Kreifelts *et al.,* 1993) is a tool developed at GMD for the management of distributed work. The users of the system share and manipulate common tasks by means of to-do lists. A task in the Task Manager is a generic structure that can be mapped onto a project, a procedure or just a shared folder, depending on the level of granularity with which it is specified. By means of the Task Manager users may organize cooperative tasks, monitor their progress, share documents and exchange messages during task performance. The system has moreover been designed for use not only over local networks, but also in a distributed environment. The architecture of the system is based on X.400 store and forward message transfer over an arbitrary number of domains. Each domain is based on a local area network and within each domain a client-server architecture is used to update a shared task structure. When users belonging to different domains are involved in a task, the task objects (and their sub-tasks) are replicated in every relevant domain. The concurrency problems that can arise from concurrent and distributed access to the tasks are managed via a set of protocols capable of preserving the causal order of operation on tasks and

resolving conflicting assignments by assuming a linear order on those events. This can sometimes cause local changes being "overwritten" by remote changes, in which case the user interface highlights and signals all the occurred changes. WebFlow and TaskManager appear to support similar functionalities through different infrastructures and protocols, namely WWW/HTTP and X.400. One relevant difference is, however, in the way inconsistencies are managed. TaskManager provides a specific component to decide which one, among a number of conflicting actions, should be accepted and which ones should be rejected. WebFlow prevents the possibility that inconsistencies ever arise through the use of the transactional capabilities provided by the CLF.

6. Conclusion

We have illustrated the WebFlow environment for distributed coordination services on the World Wide Web, focusing on the distributed workflow component. The general aim of WebFlow is to contribute to making into reality the potential for flexible IT support for organizational changes generated by the impact of the Web and the Inter/Intranet infrastructure on organizations. Further developments of WebFlow will address other aspects of IT support that are relevant for this purpose, such as electronic commerce and knowledge management tools.

Acknowledgements

We are grateful to Jean-Marc Andreoli, Hervé Gallaire and François Pacull and to the anonymous referees of the International Journal of CSCW for helpful comments on earlier drafts of this paper.

References

Alonso, G., C. Mohan, and R. Güntör (1995): Exotica/FMQM: A Persistent Message-Based Architecture for Distributed Workflow Management. In *Proceedings of the IFIP Working Conference on Information Systems for Decentralized Organizations*, Trondheim, August 1995.

Andreoli, J.-M., S. Freeman, and R. Pareschi (1996): The Coordination Language Facility: Coordination of Distributed Objects. *Theory and Practice of Object Systems*, 2:2, 1-18.

Andreoli, J.-M., H. Gallaire and R. Pareschi (1994): Rule-based Object Coordination, in *ECOOP'94 Workshop on Object-based Models and Languages for Concurrent Systems*, Springer Verlag, LNCS 924, 1994.

Bentley, R., W. Appelt, U. Busbach, E. Hinrichs, D. Kerr, K. Sikkel, J. Trevor and G. Woetzel (1997a): Basic Support for Cooperative Work on the World Wide

Web, in *International Journal of Human-Computer Studies: Special Issue on Novel Applications of the World Wide Web*, Academic Press, 1997, in press.

Bentley R., T. Horstmann, and J. Trevor (1997b): The World Wide Web as enabling technology for CSCW: The case of BSCW, in *International Journal of CSCW: special issue on CSCW and the Web*, Kluwer, 1997, in press.

Borenstein, N. (1992): Computational mail as Network Infrastructure for Computer-Supported Cooperative Work. In J. Turner and R. Kraut (eds.): *CSCW'92. Proceedings of the Conference on Computer Supported Cooperative Work*, Toronto, Canada, October 31- November 4, 1992. New York: ACM Press.

Borghoff, U. M., and J.H. Schlichter (1996a): On Combining the Knowledge of Heterogenous Information Repositories. *Journal of Universal Computer Science*. vol. 2, no. 7

Borghoff U. M., P. Bottoni, P. Mussio, and R. Pareschi (1996b): Reflective Agents for Adaptive Workflows. RXRC technical report.

Borghoff U. M., R. Pareschi, H. Karch, M. Nöhmeier, and J.H. Schlichter (1996): Constraint-Based Information Gathering for a Network Publication System. In *Proceedings of 1st International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM '96*), April 1996, London, U.K., pp. 45-59.

Bowers, J., G. Button, and W. Sharrock (1995): Workflow from Within and Without. In H. Marmolin et al. (eds.): *Proceedings of the Fourth European Conference on Computer Supported Cooperative Work (ECSCW'95)*, Stockolm, Sweden, September 10-14, 1995. Dordrecht: Kluwer Academic Publisher, pp. 51-66.

Ciborra, C. (1993): *Teams, Markets, and Systems*. Cambridge: Cambridge University Press.

DMA Technical Commitee (1995): *AIIM Document Management Alliance*. HTTP://www.aiim.org/dma/.

Dourish, P., J. Holmes, A. MacLean, P. Marqvardsen, and A. Zbyslaw (1996): Freeflow: Mediating Between Representation and Action in Workflow Systems. In *Proceedings of the ACM 1996 Conference on Computer Supported Cooperative Work*, Boston, Mass., November 16-20, 1996. New York: ACM Press.

Ellis, C., S.J. Gibbs, and G.L. Rein (1991): Groupware: Some Issues and Experiences. *Communication of the ACM* 34(1), pp. 39-58.

Ellis, C. , K. Keddara, and G. Rozenberg (1995): Dynamic Change Within Workflow Systems, *Proc. COOCS* '95, ACM Press, 1995, pp. 10-21.

Fukuyama, F. (1996): *TRUST: The Social Virtues and the Creation of Prosperity.* New York: Free Press.

Ginsburg, M., and K. Duliba (1997): Enterprise-level groupware choices: Evaluating Lotus Notes and Intranet-based solutions, in *International Journal of CSCW: special issue on CSCW and the Web*, Kluwer, 1997, in press.

Glance, N., D. Pagani, and R. Pareschi (1996): Generalized Process Structure Grammars (GPSG) for Flexible Representations of Work. In *Proceedings of the ACM 1996 Conference on Computer Supported Cooperative Work*, Boston, Mass., November 16-20, 1996. New York: ACM Press.

Kreifelts, T., E. Hinrichs, and G. Woetzel (1993): Sharing To-Do Lists with a Distributed Task Manager. In G. De Michelis et al. (eds.): *Proceedings of the Third European Conference on Computer Supported Cooperative Work (ECSCW'93)*, Milano, Italy, September 13-17, 1993. Dordrecht: Kluwer Academic Publisher, pp. 31-46.

MacLean, A., and P. Marqvardsen (1996): Crossing the Border: Document Coordination and the Integration of Processes in a Distributed Organisation. Submitted for pubblication to the *International Working Conference on Integration of Enterprise Information and Processes (IPIC '96).*

Pacull , F., A. Sandoz, and A. Schiper (1994): Duplex: A Distributed Collaborative Editing Environment in Large Scale.In *Proceedings of the ACM 1996 Conference on Computer Supported Cooperative Work*, Chapel Hill, North Carolina, October 22-26, 1994. New York: ACM Press.

Posner, I.R., and R.M. Baeker (1992): How People Write Together. In *Proceedings of the Twenty-fifth Annual Hawaii International Conference on the System Sciences*, Vol. IV, pp. 127-38.

Swenson, K. (1993): Visual Support for Reengineering Work Processes. In S. Kaplan (ed.): *Proceedings of the Conference on Organizational Computing Systems*, Milpitas, California, November 1-4, 1993. New York: ACM Press, pp. 130-141.

Suchman, L. (1987): *Plans and situated actions*. Cambridge: Cambridge University Press.

Williamson, O.E. (1975): *Market and Hierarchies*. New York: Free Press.