

# Using Java for the Coordination of Workflows in the World Wide Web

Michael Weber, Torsten Illmann  
Fakultät für Informatik, Universität Ulm

## Abstract

In this paper we introduce a workflow management system, called WebFlow, which is based on the world wide web and Java as its basic technologies. Java is used as the build time (modeling) language to define workflows as well as the implementation language for the run time workflow enactment. Due to the object-orientation of Java modular and extendible workflow types are possible. Modification of workflows is supported even at run time. Using WWW and Java eases the implementation effort of the workflow engine, since HTTP and the Java API already include functionality which needs not to be implemented anew. This is uploading and downloading of workflow applets and documents, authentication of clients, digital signing and especially the execution of workflows at the client site by the Java virtual machine. Thus, a very simple control server is sufficient, since the applets constituting the workflow coordinate themselves to a large extent. Webflow aims at application scenarios requiring flexible and modifiable workflows. It supports workflows which cross organizational boundaries, since it only relies on standard WWW mechanisms.

## 1 Introduction

Business processes which are optimized in time and flow are becoming crucial for the commercial success of companies. Such processes can be modeled as a composition of activities and subsequently mapped onto workflow management systems [1]. This phase is also called build time. During the flow of business processes the corresponding tasks constitute a so-called workflow which is controlled and coordinated by the workflow management system during run time, the second phase of workflow management. The system allocates persons to activities when the activities become active and schedules them to be performed by the individual persons. If a person receives such an activity he or she is free to choose the point in time to perform it as long as no preset time limit gets violated. If the time limit expires, the workflow management system would intervene. To perform activities a person is provided with computer-based resources, mainly documents and computer applications. These resources reside either at the person's local computer, are accessed from a server, or are sent along from person to person by the workflow management system.

A feature of most current workflow management systems is that the entire workflow is defined (modeled) before it is actually activated (instantiated). Escaping from the modeled workflow usually is impossible.

From an architectural perspective workflow management systems are mostly client/server systems. A centralized workflow engine coordinates the workflow clients being related to the workflow participants.

The growth of the internet and especially the world wide web (WWW) has caused an increasing shift from dedicated client/server software towards the exploitation of WWW

technology for many kinds of applications, also for workflow management systems. Independence of operating system and hardware platform at the client side is guaranteed when using the standard WWW protocols and document formats. The basic WWW applications still follow the client/server paradigm.

However, within the WWW and also in dedicated environments the paradigm of agent-based distributed systems is evolving. Agents are active instances which are allowed to move in the network and be executed at a location different from their original host. Agents offer advantages which ideally map to the requirements of flexible, decentralized workflow management. Agents reduce network traffic, since they execute where the resources are. I.e. in a workflow scenario the necessary control takes place at the client and not through interaction with the central server. The inherent decentralization achieves higher robustness concerning network or server failures. Agent technology allows a far-reaching decoupling of workflow clients and server.

In this paper we propose a workflow management system, called WebFlow, entirely based on the WWW and Java [2] as its basic technologies. In WebFlow Java is used as the build time (modeling) language as well as the implementation language for workflow enactment. Modular and extendible workflow definitions are possible due to the object-orientation of Java. Using WWW and Java eases the implementation effort of the workflow engine, since HTTP and the Java API already include a lot of functionality, such as uploading and downloading (HTML, HTTP), authentication (HTTP), digital signatures (JDK1.1), and execution of workflows at the client site based upon the Java virtual machine [3]. Webflow aims at application scenarios requiring flexible and run time modifiable workflows which may even cross organizational boundaries.

## **2 Requirements and Base Technologies**

Before we describe the WebFlow system in detail the core technologies being applied are sketched. Concerning workflow management we focus on the requirements such a system shall meet. Then the different design paradigms for agent systems are presented.

### **2.1 Workflow Management**

Workflow management includes a build time and a run time dimension imposing different requirements [1] each. The build time requirements comprise all issues related to modeling of control and data flow. Most of these requirements are not specific to workflow modeling, like graphical or formal representation, ease of use, readability, abstraction and modularity, or correctness. Similar requirements are found in software engineering methodologies.

The run time requirements are related to the activation, enactment and termination of workflows. The system shall be extendible to be able to handle future unforeseen application scenarios. Workflow definitions will change over time and thus workflows should be dynamically customizable and adaptable. Continuously growing application areas and increasing organizational coverage call for a scalable system. The system should be open that it can run in a distributed heterogeneous computing infrastructure.

Since in the workflow community different terminologies are used, we briefly specify the important terms being used in the paper (see also [4]).

- *Workflow*: The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.
- *Workflow definition*: A workflow definition identifies the various activities, procedural rules and associated control data used to manage the workflow during process enactment.
- *Activity*: One logical step in a workflow definition. An activity cannot be decomposed any further.
- *Composite workflow*: A workflow definition which contains activities and/or again composite workflows.
- *Workflow instance*: An instantiated workflow definition which is in some execution state (ready, executing, paused). Several instances can be derived from the same workflow definition. Workflow instances are managed by the workflow management system.
- *Subworkflow, superworkflow*: Subworkflows are used for refinement within other workflows, just as a subroutine in programming languages. Superworkflows comprise other workflows.
- *Workflow participant*: A system representation of a user which is related to the execution of an activity. This can be a representation of a human user or of a program which we call auto-user.

## 2.2 Agent technologies

The dominating design paradigm for distributed applications still is the client/server paradigm. Also most of the workflow management systems are implemented using this approach. With the upcoming need for network-centric computing characterized by the huge increase of internet usage and applications agent-based paradigms are evolving.

They promise to achieve much more dynamics, higher scalability, reduced network traffic, and failure tolerance through autonomy than it is possible with the client/server approach. Carzaniga et al. identify three main alternatives of agent paradigms and technologies [5].

- *Remote evaluation*: A requestor has the know-how (the code) and sends this to an executor which provides processor and I/O-resources. I.e. the client comes to the server. This paradigm is implemented in the WWW with Java servlets [6].
- *Code on demand*: A requestor has the I/O-resources and the processor. It receives the code on demand from a provider. I.e. the server comes to the client. The code is executed in the host environment until it terminates. This paradigm is implemented in the WWW using Java applets [7].
- *Mobile agent*: An agent moves to a location where the processing and I/O-resources are which it will use there on behalf of a requestor (the one sending out the agent). Mobile agents can move further on their own will without returning to the requestor. I.e. they take

code and execution state with them. Odyssey [8] or aglets [9] are implementations of this paradigm.

Webflow uses the code on demand paradigm where each activity is represented by a Java applet. Opposed to most applets different Webflow applets can coordinate themselves by communicating through a shared data space.

### 3 Modeling of Workflows

Workflows are modeled in Java instead of using an extra definition language. This provides certain advantages:

- Workflows are defined in an object-oriented way and therefore they can be easily reused, extended and adapted to re-engineered workflows.
- There is no interpreter needed to execute workflow instances. The Java Virtual Machine (JVM) will do the interpretation.

In WebFlow, abstract workflow classes are provided which already implement the general execution procedure of a workflow. A new workflow is defined by inheriting the abstract class and implementing abstract methods defining the incoming (`initDataflow`) and outgoing data flow (`putDataflow`), the outgoing control flow (`putControlflow`) and the initialization of an application to be used for the activity (`initApplication`). The abstract methods of an activity are:

```
abstract class Workflow {
    void initApplication(Application a);
    void initDataflow();
    void putDataflow();
    void putControlflow();
    ... }
```

Jablonski and Bussler define different perspectives of a workflow [1]. The following paragraphs shortly explain how they are expressed in WebFlow.

**Functional Perspective.** Workflows are being developed *hierarchically*. A modeler may specify an *activity* or a *composite* workflow. While an activity is used to perform a set of simple work items for one user, a composite workflow contains several subworkflows (see figure 1).

A composite workflow contains at least two activities: a *start* and an *end* activity (see also figure 2). The start activity is called first. After its execution, subworkflows can be defined and optionally activated. When all subworkflows have been finished, the end activity is started. Its task is to evaluate the results of the subworkflows, perform hierarchical data flow and coordinate workflows which are concurrent to this composite workflow. The analogy in software development is a *divide & conquer* principle. The main routine (composite workflow) divides the problem into several subproblems (start), calls a set of subroutines to solve the subproblems (subworkflows) and finally combines the subresults to the total result (end).

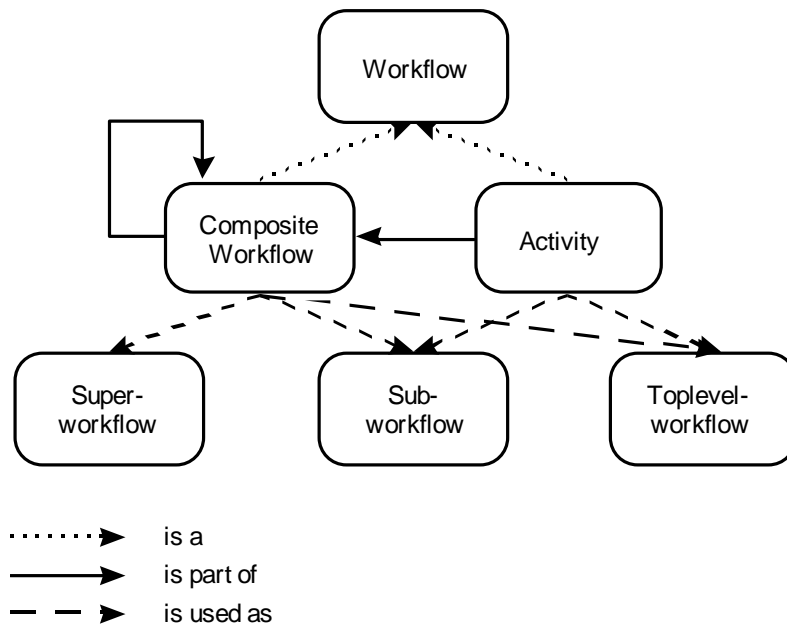


Figure 1: Relationship diagram of workflows

Defining subworkflows is done within the method `defineSubworkflows`. For instance, to specify two subworkflows „flow1“ and „flow2“ the following code is required:

```

void defineSubworkflows(Workflows subflows) {
    subflows.add("flow1", "MyWorkflow1.class");
    subflows.add("flow2", "MyWorkflow2.class");
}
  
```

„Flow1“ and „flow2“ are so-called workflow variables and „MyWorkflow1.class“ and „MyWorkflow2.class“ are the names of the real Java classes. This abstraction enables to easily change a workflow, even at run time, without modifying every subworkflow referring to it.

**Operational Perspective.** One main task of workflow management is the coordination of applications. In WebFlow, an activity refers to a standard application class providing a set of standard work items. These work items include reporting messages, handling dialogs, accessing databases via JDBC, uploading, downloading and modifying documents, e-mailing, etc.

To integrate external applications, the standard application class has to be inherited and the `execute` method to be overwritten. This method represents the interface from workflow to application. E.g.: the start of a local word processing program with a certain document could be implemented as follows:

```

class LocalApplication extends Application {
    Object execute (Object data) {
        String filename = (String) data;
        Runtime rt = Runtime.getRuntime();
        Process p = rt.exec("WordProcessing" + filename);
        return p;
    }
}
  
```

The `execute` method is called automatically after starting a workflow and enables data flow between workflow and application. Within it, one implements the access to arbitrary applications. Java easily enables the access of remote application through common protocols like RMI, RPC, CORBA and JDBC. The restriction that remote applications have to reside on the host computer is eliminated when using JDK 1.1 and trusted applets. Access to local applications also requires this feature.

**Behavioral Perspective.** To coordinate the control flow of concurrent workflows, three low-level primitives are provided:

- `insertLink`    activates a workflow instance for a certain user and inserts a link into his work-to-do-list (optionally specifying a time constraint).
- `removeLink`    finishes a workflow instance and removes the corresponding link from the user's work-to-do-list.
- `updateLink`    updates a workflow instance, i.e. changes user, workflow class, application class, document or time constraint.

These primitives enable to define more complex and reusable control flow primitives like serialization, parallelism, alternative execution, and joining of parallel workflows. For example, an alternative execution may be defined as follows:

```
class MyControlFlow(Workflow w, boolean cond, Link l1, Link l2) {
    if (cond == true)
        w.insertLink(l1);
    else
        w.insertLink(l2);
}
```

Since Java is an object-oriented programming language, there are no limitations to modularly designing complex high-level control flow constructs.

**Informational Perspective.** Since the different views of data flow and control flow often confuse, a simple method for exchanging data among workflows has been chosen in WebFlow. Figure 2 illustrates the basic idea. All concurrent workflows can read and write to a shared database. The workflow class supplies the methods `putData` and `getData` to enable this agent-based data flow. To perform hierarchical data flow, a composite workflow's start and end activity is allowed to access the database of subworkflows as well as the one of concurrent workflows. The dashed lines in figure 2 show a hierarchical data flow from subworkflows to concurrent workflows.

**Organizational Perspective.** The organizational perspective is designed simple and flexible. Each composite workflow has to define and assign the workflow participants being available for its subworkflows. Since definition and assignment of workflow participants is done after the execution of the start activity, the actual assignment can be done in a fixed or in a dynamic way (i.e. by querying a user's database or by prompting for them in the start activity). Subworkflows use workflow participants to coordinate workflows. This provides easy maintenance when workflow participant assignments change and offers the possibility to change workflow participants even during run time (see section 3.2).

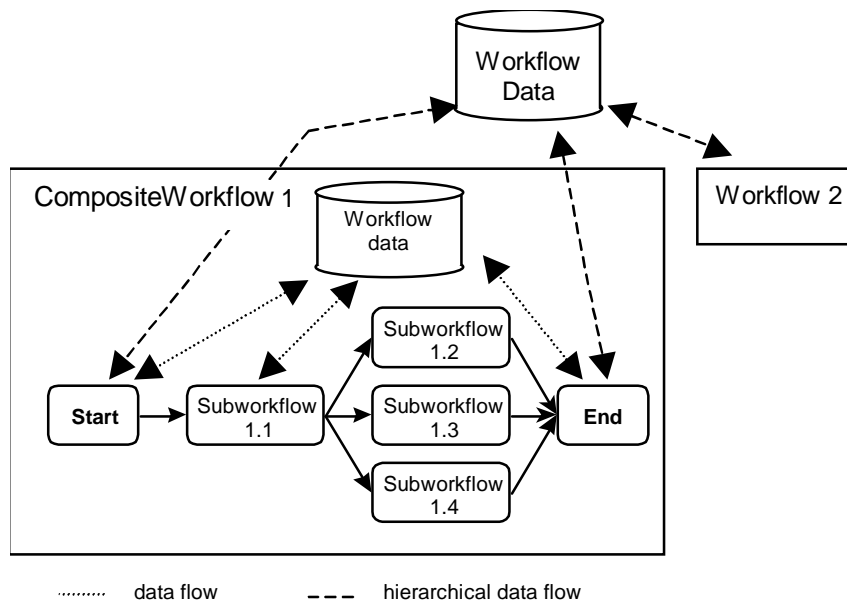


Figure 2: Data flow of composite workflows

### 3.1 Application Example

An example of a simple business process is explained in this section to illustrate the usability of WebFlow. Figure 3 illustrates the composite workflow „purchasing new computer equipment“ which could happen in a company. It involves five workflow participants: a manager, a computer specialist, a financial specialist, a secretary and an auto-user. The scenario is as follows. The manager realizes that new computer equipment is needed. He contacts the computer specialist to get informed about possible shops and cost. He simultaneously asks the financial specialist to look for the current budget being available. If the budget is higher than the cost, the secretary will order the equipment from the chosen shop. The manager gets the message of a successfully given order. Otherwise, if the budget is too low, he gets an appropriate trouble message. Joining the parallel tasks of the specialists and the decision how to continue, is defined as an auto-user activity, i.e. it is executed automatically.

The overall composite workflow contains the start activity, the end activity and four other activities as subworkflows. Hence, the definition of four inner subworkflows reads as follows:

```
void defineSubworkflows(Workflows subflows) {
    subflows.add("budget", "GetBudget.class");
    subflows.add("offer", "GetOffer.class");
    subflows.add("decision", "Decision.class");
    subflows.add("order", "DoOrder.class");
}
```

Defining the workflow participants does not require the definition of the manager and the auto-user. The manager only acts in the start and end activities of the composite workflow and thus has to be defined in the superworkflow. The auto-user is a predefined workflow participant. Each workflow participant is represented by an agent which is similar to

representing him by his role. From the Java perspective this means representing participants through variables which take the participant as its value.

```
void defineSubagents(Agents agents) {
    agents.add("computer-spec", "Jack");
    agents.add("financial-spec", "Steve");
    agents.add("secretary", "Susan");
}
```

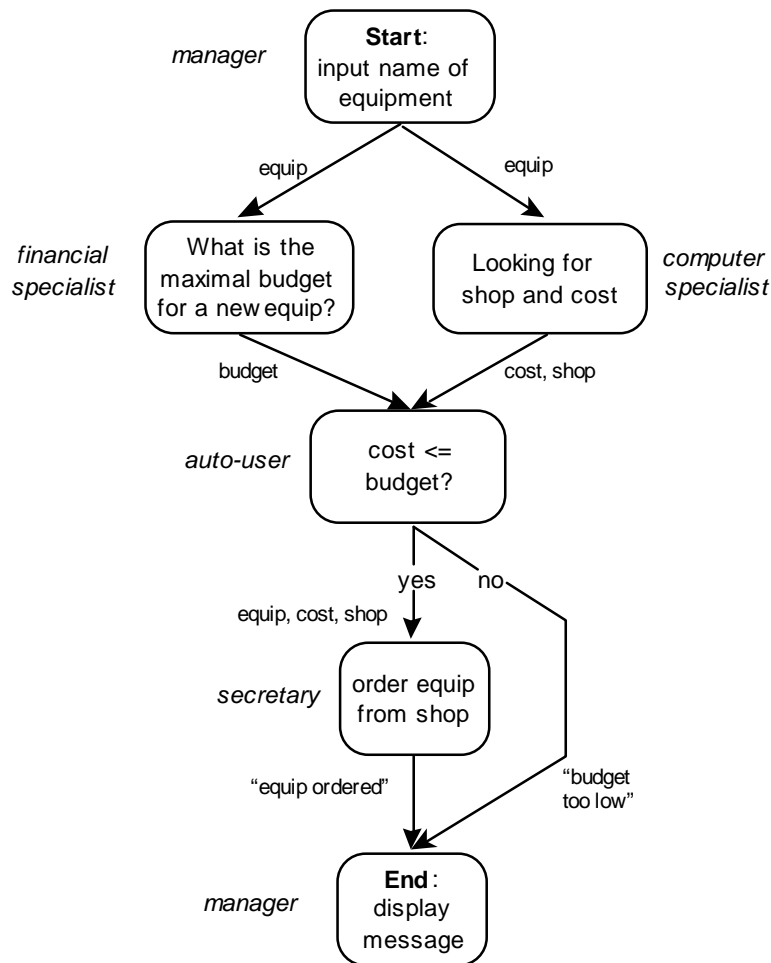


Figure 3: A sample composite workflow: "purchasing new computer equipment"

The start activity has to prompt for the required name of the equipment. The name is passed to the subworkflows within the `putDataflowStart` method. The subworkflows are activated using the `putControlflowStart` method.

```
void putDataflowStart(Dataflow subflows) {
    subflows.putData("equip", <NameOfEquipment>);
}

void putControlflowStart(Controlflow sub..) {
    subflows.insertLink("computer-spec", "offer");
}
```

```

        subflows.insertLink("financial-spec", "budget");
    }

```

The activity of the computer specialist has to receive as input the name of the equipment and it has to output its cost and the shop where it should be ordered. It starts the „decision“ activity if necessary.

```

class GetOffer extends Workflow {
    String equip;
    ..
    void initDataflow() {
        equip = getData("equip").toString();
    }
    void putDataflow() {
        putData("shop", <NameOfChosenShop>");
        putData("cost", <CostsOfEquipment>");
    }
    void putControlflow() {
        if (!state.isActive("Budget"))
            insertLink("auto", "decision");
    }
}

```

The „decision“ activity requires the budget and cost information. Depending on whether the budget is sufficient, it activates the „order“ activity or it finishes the superworkflow with outputting a failure message to the manager. Here is the code:

```

class Decision {
    int budget, cost;
    ..
    void initDataflow() {
        budget = getData("budget").toInt();
        cost = getData("cost").toInt();
    }

    void putControlflow() {
        if (cost < budget)
            insertLink("secretary", "order");
        else
            // activate end of
superworkflow
            updateLink("super", "super");
    }

    void putDataflow() {
        if (cost > budget)
            putData("message", "budget too low");
    }
}

```

The other activities are defined in analogy to the presented ones.

### 3.2 Modification of Workflows

The modification of workflows is an important requirement of flexible workflow management systems, because business processes are being re-engineered permanently. In general, two

different aspects may be considered. On the one hand workflow definitions can be modified, on the other hand workflow instances.

As explained previously workflow definitions are Java classes. There are two ways of modifying Java classes. Either the code is changed or it is inherited and extended. Both possibilities have advantages and disadvantages. The direct change of the source code has the advantage that all active workflow instances are dynamically being updated, too. The disadvantage is that the old workflow definition is lost and inconsistencies may be introduced easily, if either control or data flow has been changed. The more recommendable way is to inherit and extend the class. Thus, valuable parts of the old definition can be obtained and new functionality added, which allows a modular design of workflow definitions. The old definition still exists and workflow instances of both definitions can be created. The problem is that active instances are not being changed on the fly this way. The following paragraph describes a solution to this problem.

Workflow instances are instantiated workflow definitions. Related to WebFlow, instances are parameterized Java classes together with data and control data required by the instance. The instance's data consists of the names of the workflow and of the application class, the URL of the document, the user executing it, and the time constraints. The control data consists of the actual state, a unique id, the names of concurrent workflows and the concurrent workflow participants. As a special feature of WebFlow a composite workflow currently running subworkflows can control and influence the execution of these subworkflows. Workflow operations can be placed on them to change their state, i.e. (re-)activate, abort, pause, or resume them. In addition, the accompanying document, the workflow class, or the application may be exchanged on the fly. It is possible to reassign workflow participants which makes sense when users suddenly get ill or are on vacation, or when a role concept should be incorporated. These features are possible, since subworkflows are defined through workflow variables and workflow participants through agents.

## **4 Implementation Architecture**

The key idea of the WebFlow system is the realization of workflows with Java applets. Since Java applets are locally executed programs loaded over the network together with an HTML document, they are an ideal concept for decentralized, agent-based and self-coordinating workflows. An applet is able to perform certain activities for the user (i.e. on a document), to call local and remote applications and to contact the workflow engine to invoke the subsequent control flow.

The main components of the WebFlow system are shown in figure 4. The system consists of the WebFlow engine, a database management system and a web server on the server machine. The client side comprises a web browser and local applications. Remote applications on third-party computers can be invoked as well.

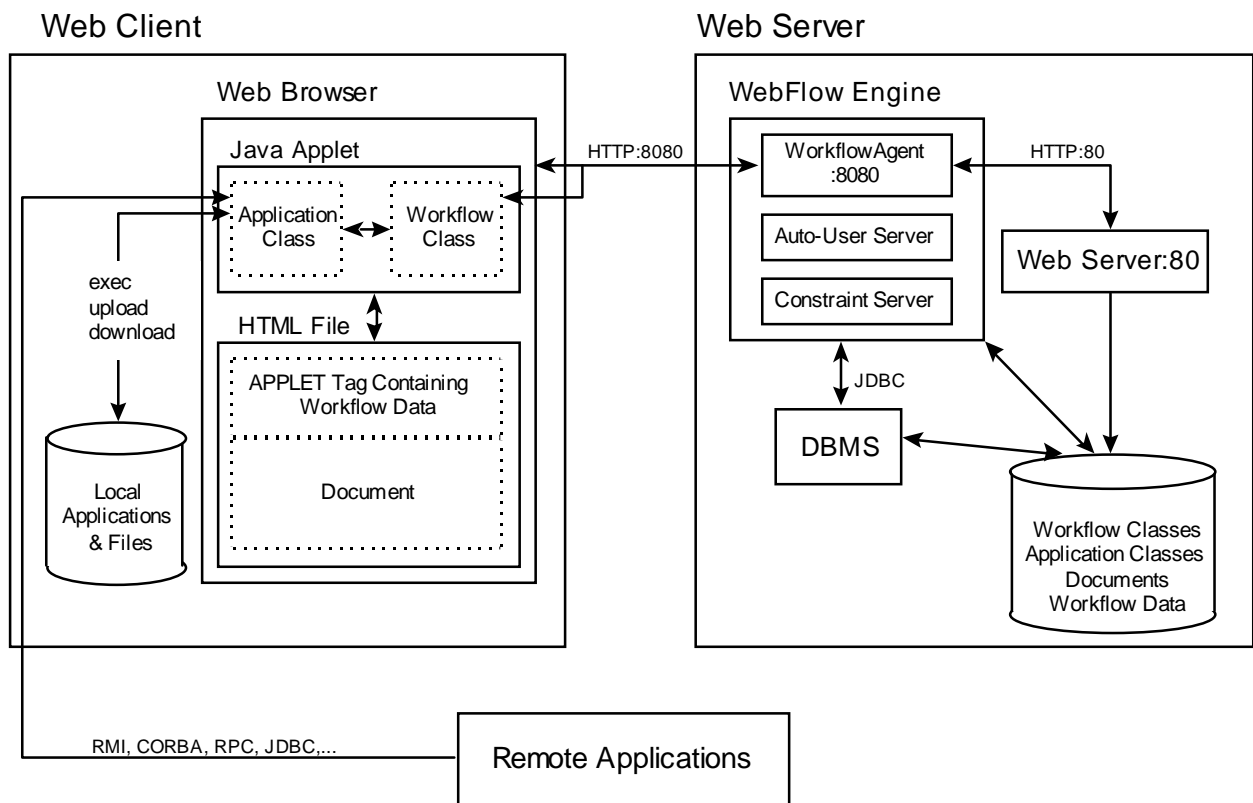


Figure 4: Architecture of the WebFlow system

To meet the quality factors and requirements of a workflow management system three major design considerations have been taken into account for the WebFlow system.

- *Decentralization*  
Each instance of a workflow activity is a self-sufficient program being executed on the client machine. The workflow engine does not act as a central control entity but executes commands of activity instances being active at client machines. The engine is only activated twice, once before starting and once after finishing an activity instance. In the first case, it supplies the activity instance with required data, in the later, it executes the control and data flow statements of the activity instance.
- *Self-Coordination*  
An activity runs independent from the engine. When finishing, it coordinates the subsequent control and data flow, i.e. it activates other activities, stops concurrent activities, or passes information to them.

The execution of a workflow instance proceeds as follows.

- The user willing to start a workflow loads his *work-to-do-list* into the web browser. The work-to-do-list is a simple HTML file containing descriptions of active workflow instances and HTML links (URLs) pointing to the instances themselves which are again HTML documents.

- Activating a link means loading the document and starting the corresponding workflow instance. The web browser does not communicate with the web server directly but through the *workflow agent* which is part of the WebFlow engine. The workflow agent pipes HTTP requests from the web browser to the web server and HTTP replies the opposite way. Whenever a workflow document is requested, the workflow agent gets the document from the server and attaches the corresponding workflow applet before transferring it to the browser. Attaching an applet means inserting an HTML APPLET tag to the document stream. Required workflow data is added to the APPLET tag as parameters.
- The workflow applet is executed automatically on the client machine in the web browser's environment. There, it may perform certain standard tasks like executing a dialog with the user, calling local or remote applications, up- or downloading of documents, sending an email or coordinating the execution of concurrent workflow instances (these are workflow instances belonging to the same superworkflow).
- With finishing the work items of the current activity, the workflow applet contacts the workflow engine to perform a transaction of certain control- and dataflow statements coordinating the concurrent workflows. During this (short) execution of that transaction no concurrent workflow is allowed to contact the engine.

This design meets the following quality factors for workflow management systems:

- *Scalability*  
The agent-based style of workflow management in WebFlow is scalable, since the workflow instances are distributed on the client machines and self-coordinating. The engine simply executes commands from the workflow instances without caring about the overall process. Thus the instances on the client machines are the active part in WebFlow, the engine is passive.
- *Robustness and Correctness*  
Temporary failures of computers or the network might only cause the repetition of the lost activity in the worst case. Workflow instance and engine only connect when finishing an activity and all commands are being transmitted within a transaction. Therefore, inconsistencies do not happen due to the all-or-nothing semantics of our transactions.
- *Portability*  
Workflows and the engine may be executed on arbitrary computers and operating systems, since they are implemented entirely in Java.
- *Ease of Deployment*  
The client machines only require the installation of a web browser. Since the engine is also implemented in Java, it is portable and can be installed on the web server's machine. It communicates with every browser and server as long as they understand HTTP. Existing local and remote applications may be integrated through common protocols like RMI, RPC, CORBA, JDBC, ODBC, etc.
- *User-Friendliness*  
Dealing with workflows is simple, because they run in the user's favorite web browser and contain Java interactive components and HTML documents which have developed a standard already. The look-and-feel is similar on different machines.

- *Reusability, Extendibility and Adaptation*

Workflow definitions are being modeled in an object-oriented and hierarchical way in Java. By inheriting workflow definitions, they can be easily extended, reused, or adapted to re-engineered business processes. Workflow instances can be dynamically adapted at run time.

#### 4.1 Engine Architecture

One design objective of the WebFlow engine was to place as much intelligence as possible to the decentral, agent-based Java workflows and keep the intervention of the (central) engine as low as possible. Nevertheless, the engine still has to control the following tasks:

- dynamically creating work-to-do-lists when requested,
- authenticating users,
- transmitting workflow instances including actual workflow data, documents and applications to the clients.
- processing and replying to workflow requests such as `insertLink`, `removeLink`, `updateLink`, `getData`, `putData`,
- controlling time constraints of workflow instances,
- running workflows that should be executed automatically, i.e. workflows which do not need a human workflow participant.

## 5 WebFlow Extensions

The current WebFlow implementation is in a prototype state. The engine is fully operational as being described. Experimentations with application examples using business processes taken from the university environment have been performed as a proof of concept. These test have shown a couple of future extension possibilities.

The release of JDK1.1 [10] offers many new features most of them due to the security API and its feature of trusted applets. They are:

- digitally signing of documents (security API),
- access of local and third-party applications and databases (trusted applets),
- integrating Java Beans as workflow applications (BDK, reflection API),
- reducing transmission time of workflow applets with JAR files,
- printing of documents (improved awt API),
- uploading and downloading files without the help of the web browser (trusted applets).

The authentication scheme actually is the BASIC scheme (proposed for HTTP 1.1) which is neither a secure method of user authentication nor does it protect the transmitted entity [11]. Future versions of HTTP will provide more secure authentication schemes, e.g. digest authentication. Another possibility would be the use of *Secure HTTP*, *HTTP 2.0* or *SSL* which

additionally guarantee a secure transmission. Such security measures are required when WebFlow shall be used in an intranet fashion within the open internet. When workflows cross organizational borders security is also a big concern.

Furthermore, our current constraint model of workflows could be improved. In the actual design, only time constraints (start date, end date, maximal duration) can be specified. A more sophisticated model requires arbitrary constraint conditions depending on time, workflow participants and workflow data.

Future work will also include the development of a Java-based, visual and object-oriented process flow editor, which allows to define workflows graphically. It should be possible to create, move and modify workflows and their the control and data flow with the mouse in a drag&drop manner. One should be able to zoom hierarchically into and out of composite workflows, e.g. by double-clicking the corresponding GUI component. In addition, every GUI component representing a workflow should provide a pop-up menu which allows to edit the properties of the workflow such as the accompanying document, application, workflow participants and constraints. The corresponding Java classes will be generated automatically.

Such an editor will push down Java from being the modeling language, as described in the paper, to an intermediate language in WebFlow. We do not see this as being contradictory to the concepts we presented. Java can still be used as a modeling language for sophisticated workflow designers. Graphical editors on top allow to have customized editors for specific application domains or for specific modeling methodologies, such as petri nets or state charts.

## 6 Related Work

Mobility, decentralization, agent and WWW technology have been applied to workflow management also in other systems to certain extents. We cover here a range of example systems related to the WebFlow design issues.

**Mobile workflows.** ProMinanD [12] follows the paradigm of migrating office folders, called electronic circulation folders (ECF). Each ECF contains in its header part among other information its migration specification. The second part holds the activity data. Since ECFs contain their routing information, they can be compared to agents migrating through a workflow management system. Local migration servers (LMS) running on each machine serve as the local run time environments for ECFs. Migration from LMS to LMS is handled by global migration servers. However, ProMinanD is based on distributed database technology and does not use any of the agent paradigms being described earlier.

**WWW-based workflow clients.** Many commercial and research workflow management systems use WWW browsers as a front end towards the user, e.g. [13]. As an example WWWorkflow is briefly described here. WWWorkflow [14] is designed for intranets where the users exploit WWW browsers as the only client software. WWWorkflow's architecture contains three main parts: the workflow engine, the workflow database and a CGI-gateway to the WWW. The workflow engine manages the control flow of active workflow instances in the system. It understands a „workflow language“ (WFL) being a dialect of Tcl [15] augmented by commands to create, control and modify workflow instances. These workflow instances together with their current status are stored in the workflow database. The workflow

engine operates on these database entries. In the database also the workflow definitions are held. Modeling constructs include steps (i.e. activities), step iterations (i.e. simple composite workflows), priorities, activation conditions and timing requirements. A common gateway interface (CGI) front end allows users to interact with the system through a triple frame to-do-list presented in their browsers. A first frame displays the actual to-do-list of a user. A middle frame presents the workspace containing a HTML document describing the activity. In the bottom frame buttons allow to submit the completion status of this activity.

**Workflow agents.** Dartflow [16] is a workflow management system based on mobile agents. It uses so-called process-agents which carry data and control flow of a specific workflow instance with them. Whenever an activity in the workflow instance is related to a specific machine, the process-agent migrates to this machine to perform the required actions locally, e.g. to access a local database. Process-agents are controlled by three types of agent servers: the organization server, the tracking server and the worklist server. The organization server contains generic workflow definitions and provides process-agents with the current organizational information while they are active. The organization server thus implements up-to-date role to person allocation and through this actualized routing information. Before a process-agent starts its actual workflow instance it registers at the tracking server which keeps track of all alive process-agents and their sequence of activities. The other agent servers notify the tracking server of any changes which takes appropriate action that the affected process-agents are notified before carrying out the next activity. This is possible, since all process-agents contact the worklist server after each commenced activity to add themselves to the worklist of the next user. These worklists are displayed to the users in their WWW-browser using Java applets. The work items are given as forms being handled by CGI scripts. The scripts are directly related to a process-agent responsible for the entire workflow which contains this current activity.

**Decentralization.** Other approaches to decentralize workflow management are also found in literature. They either build on distributed processing environments, like CodAlf [17] or BPAFrame [18], or they are based on message passing, like Exotica [19].

## 7 Conclusions

WebFlow is a workflow management system based on the WWW and Java as its basic technologies. Java is used as the build time (modeling) language as well as the implementation language for workflow enactment. In WebFlow modular and extendible workflow definitions are possible due to the object-orientation of Java. Inheritance of already modeled workflow definitions allows an easy construction of sophisticated workflows keeping the modeling process still manageable. Modification of workflows is possible even at run time.

Using WWW and Java eases the implementation effort of the workflow engine, since HTTP and the Java API already include functionality which has not to be implemented anew. This is uploading and downloading of workflow applets and documents, authentication of clients, digital signing and especially the execution of workflows at the client site by the Java virtual machine. I.e. a very simple control server is sufficient, since the applets constituting the workflow coordinate themselves to a large extent.

## 8 References

- [1] S. Jablonski, C. Bussler: Workflow Management Modeling Concepts, Architecture and Implementation, London, International Thomson Computer Press, 1996.
- [2] K. Arnold, J. Gosling: The Java Programming Language. ACM Press Books, Addison Wesley Longman, 1996.
- [3] T. Lindholm, F. Yellin: The Java Virtual Machine Specification. ACM Press Books, Addison Wesley Longman, 1996.
- [4] Workflow Management Coalition: Terminology & Glossary. Document No. WFMC-TC-1011, Issue 2.0, June, 1996.
- [5] A. Carzaniga, G. Picco, G. Vigna: Designing Distributed Applications using the Mobile Code Paradigms. International Conference on Software Engineering, 1997.
- [6] Sun Microsystems: The Java Servlet API White Paper. Technical report, Sun Microsystems, available from <http://jserv.javasoft.com/products/java-server/webserver/beta1.0/doc/index.html>, 1997.
- [7] D. Flanagan: Java in a Nutshell. O'Reilly, 1996.
- [8] J. White: Mobile Agents White Paper, available from <http://www.genmagic.com/agents/Whitepaper/whitepaper.html>, 1996.
- [9] D. Lange: Programming Mobile Agents in Java - A White Paper. IBM Corp., available from <http://www.ibm.co.jp/trl/aglets/whitepaper.htm>, 1996.
- [10] Sun Microsystems: The Java Development Kit 1.1 Documentation. Technical report, Sun Microsystems, available from <http://java.sun.com:80/products/jdk/1.1/docs/index.html>, 1996.
- [11] HTTP Working Group: Hypertext Transfer Protocol - HTTP/1.1, Internet Draft, <draft-ietf-http-v11-spec-07>, 1996.
- [12] B. Karbe, N. Ramsperger, P. Weiss: Support of Cooperative Work by Electronic Circulation Folders. Conference on Office Information Systems, 1990.
- [13] C. Ellis, C. Maltzahn: The Chautauqua Workflow System. Hawaii International Conference on System Sciences, 1997.
- [14] C. Ames, S. Burleigh, S. Mitchell: WWWorkflow: World Wide Web based Workflow. Hawaii International Conference on System Sciences, 1997.
- [15] J. Ousterhout: Tcl and the Tk Toolkit. Addison-Wesley, 1994.
- [16] T. Cai, P. Gloor, S. Nog: DartFlow: A Workflow Management System on the Web using Transportable Agents. Technical Report PCS-TR96-283, Dartmouth College, 1996.
- [17] A. Schill, C. Mittasch: CodAlf: A Decentralized Workflow Management System on Top of OSF DCE and DC++. 3rd International Symposium on Autonomous Decentralized Systems, 1997.
- [18] C. Mittasch et al.: Design and Use of BPAFrame - a Decentralized CORBA-based Workflow Management System. World Computer Congress, 1996.
- [19] G. Alonso, D. Agrawal, A. Abbadi, K. Kamath, R. Günthör, C. Mohan: Exotica/FMQM: A Persistent Message-Based Architecture for Distributed Workflow Management. IFIP Working Conference on Information Systems for Decentralized Organizations, 1995.

### Addresses of Authors

Prof. Dr. Michael Weber  
Department for Distributed Systems  
University of Ulm  
D-89069 Ulm  
weber@informatik.uni-ulm.de

Torsten Illmann  
Department for Distributed Systems  
University of Ulm  
D-89069 Ulm  
tillmann@hydra.informatik.uni-ulm.de