

# WEBFLOW: DECENTRALIZED WORKFLOW MANAGEMENT IN THE WORLD WIDE WEB

MICHAEL WEBER, TORSTEN ILLMANN, ALBRECHT SCHMIDT  
Department for Distributed Systems  
University of Ulm, Germany

## 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.

## KEYWORDS

workflow management, decentralization, self-coordination

## 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. To perform activities a person is provided with computer-based resources. A feature of most current workflow management systems is that the entire workflow is defined before it is actually activated. Escaping from the modeled workflow usually is impossi-

ble. 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 world wide web (WWW) has caused an increasing shift towards the exploitation of its 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 WWW protocols and document formats. Additionally the paradigm of agent-based distributed systems is evolving. Agents are active instances which are allowed to be executed at a location different from their original host. When using agents in a workflow scenario the necessary control takes place at the client and not through interaction with the central server. The inherent decentralization achieves a far-reaching decoupling of workflow clients and server.

We propose a workflow management system, called WebFlow, entirely based on the WWW and Java [2] as its basic technologies. Java is used as the build time language as well as the implementation language for workflow enactment. Modular and extendible workflow definitions are possible due to the object-orientation of Java. Webflow aims at application scenarios requiring flexible and run time modifiable workflows which may cross organizational boundaries.

## 2 MODELING OF WORKFLOWS

Workflows are modeled in Java instead of using an extra definition language. Thus workflows are defined in an object-oriented way and maybe easily reused, extended and adapted to re-engineered workflows. No interpreter is required to execute workflow instances. The Java Virtual Machine [3] 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 (`putCon-`

ontrolflow) 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();
    ... }
```

Workflows are being developed hierarchically. A modeler may specify a single activity or a composite workflow. A composite workflow contains at least two activities: the start and the end activity. After executing the start activity, subworkflows can be defined and optionally activated. When all subworkflows have been finished, the end activity is started in order to evaluate the results of the subworkflows, perform hierarchical data flow and coordinate workflows which are concurrent to this workflow. 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 actual Java classes. This abstraction enables to change a workflow even at run time without modifying every subworkflow referring to it.

One main task of workflow management is the coordination of applications. In WebFlow, an activity refers to an 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 application class has to be inherited and the execute method to be overwritten. 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. Java enables the access of remote applications

through protocols like RMI, RPC, CORBA and JDBC.

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. 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 control flow primitives like serialization, parallelism, alternative execution, and joining of parallel workflows.

All concurrent workflows can read and write to a shared database. The workflow class supplies the methods putData and getData to enable this 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.

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.

## 2.1 MODIFICATION OF WORKFLOWS

The modification of workflows is an important requirement of flexible workflow management systems. Two different aspects may be considered. On the one hand workflow definitions can be modified, on the other hand workflow instances.

Workflow definitions are Java classes which can be modified in two ways. Either the code is changed or it is inherited and extended. The direct change of the source code has the advantage that all active workflow instances are dynamically updated, too. The disadvantage is that the old workflow definition is lost and inconsistencies may be introduced. The recommendable way is to inherit and extend the class. Thus, valuable parts of the old definition can be obtained and new functionality added. The old definition still exists and workflow instances of both definitions can be created. However, active instances are not changed on the fly this way. The following paragraph describes a solution to this problem.

Workflow instances in WebFlow are parameterized Java classes. The instance's data consists of the names of the workflow and the application class, the document URL, the user executing it, and time constraints. The control data consists of the actual state, a unique id, the names of concurrent workflows and the concurrent workflow participants. A workflow currently running subworkflows can

control and influence the execution of these subworkflows. Workflow operations can 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. These features are possible, since subworkflows are defined through workflow variables and workflow participants through agents.

### 3 IMPLEMENTATION ARCHITECTURE

The key idea of WebFlow 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, self-coordinating workflows. Each applet in WebFlow is able to call local and remote applications and to contact the workflow engine to invoke the subsequent control flow.

The WebFlow system (figure 1) 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.

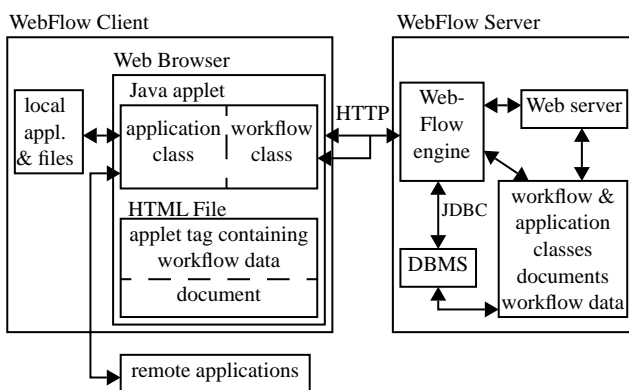


Figure 1: Architecture of the WebFlow system

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 HTML file containing descriptions of active workflow instances and links (URLs) pointing to the instances themselves which are again HTML documents. Activating a link means to load the document and start the corresponding workflow instance. The web browser does not communicate with the web server directly but through the workflow agent, a 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 by 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 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 belonging to the same superworkflow. With finishing the work items, the workflow applet contacts the workflow engine to perform a transaction of control- and dataflow statements coordinating the concurrent workflows.

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 has to do the following tasks: dynamically create work-to-do-lists when requested, authenticate users, transmit workflow instances to the clients, process and reply to workflow requests, control time constraints, and run workflows that should be executed automatically by a non-human participant.

### 4 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, such as uploading and downloading of workflow applets and documents, authentication of clients, application of digital signatures and especially the execution of workflows at the client site by the Java virtual machine. I.e. a simple control server is sufficient, since the applets constituting the workflow coordinate themselves to a large extent.

### 5 REFERENCES

- [1] S. Jablonski and C. Bussler, *Workflow Management Modeling Concepts, Architecture and Implementation* (London: International Thomson Computer Press, 1996).
- [2] K. Arnold and J. Gosling, *The Java Programming Language* (New York: ACM Press Books, 1996).
- [3] T. Lindholm and F. Yellin, *The Java Virtual Machine Specification* (New York: ACM Press Books, 1996).