Adding Distribution to a Workflow Management System

Laszlo Böszörményi
Institute of Information Technology
University of Klagenfurt, Austria
laszlo@itec.uni-klu.ac.at

Robert Eisner, Herbert Groiss
Groiss Informatics GesmbH
Klagenfurt, Austria
herb@groiss.com

Abstract

The Workflow Management System AltaVista Works has been extended to be able to process workflows in a distributed, partly even parallel manner. Build-time data are replicated over all participating workflow servers. The distribution of workflow execution is based on migration. Instead of the usual static pre-partitioning, a highly dynamic management of distribution is applied.

1 Introduction

Workflow-Management is the automation of business processes (BP). The participants of a BP execute coordinated steps in a distributed environment, via the exchange of documents, in order to achieve a common business aim [11, 6].

These steps can be executed by different processing units (typically humans and computers in cooperation), who may play different roles in the process. A given step can be executed by a processing unit playing the desired role, the identity of the unit is irrelevant.

A Workflow Management System (WfMS) is a collection of all facilities that are needed to define, manage and execute workflows [6]. A typical WfMS supports the following three functional areas [6]:

1. **Build-Time Functionality.** Support for the definition and modeling of workflow processes. The structural aspects of the organization, such as users and roles are managed here as well.

2. **Run-Time Control Functionality.** Management of the workflows during execution, embedded in a corresponding execution environment. This involves assuring proper order of execution steps.

3. **Run-Time Interaction Functionality.** Management of the interaction with human users and other IT systems.

Early WfMSs concentrated on small groups of users in a centralized environment. With the growing acceptance of workflows, the technology is used more and more in large organizations, which generate a high load and which are inherently distributed. These two new aspects leads to the desire to turn existing WfMSs into distributed ones (DWfMS).

In this paper the design of the DWfMS AltaVista Works is presented. Before going into details, the general requirements on distributed workflow execution must be regarded.

2 Distribution in workflow systems

The following basic questions must be discussed: How are the build-time data stored and accessed? How is the distributed execution controlled? How are the execution steps distributed?

2.1 Build-time data

It is a basic precondition that build-time data are accessible to all participants of a distributed workflow execution.

In the course of dynamic task assignment, the WMS selects an appropriate execution unit for a certain role. For this action it needs the definition of a given workflow as well as the roles, users, and departments defined in the workflow cluster.

How should we achieve this global view on build-time data?

One possibility is to distribute the data over all participants, and provide transparent access via a corresponding directory service. As build-time data are used quite heavily, workflow servers accessing data located at remote servers may imply a high load on the network.

Another solution could be the use of a central server for build-time data. Such an approach has the usual disadvantages of centralized algorithms: poor scalability and single point of failure at the server.

A much more promising approach is data replication. As build-time data do not change too often and their size is also moderate, this seems to be the most advantageous approach.
This solution is used e.g. in Exotica with clusters [1], and this is our approach as well.

2.2 Kinds of distributed execution

For distributed execution of workflows, we have two choices: Either the server that started the workflow controls the remote execution of individual steps centrally, or the workflows migrate from one server to the other, carrying its execution state with.

2.2.1 Remote execution. With this approach the server that creates and starts a workflow remains responsible for it for its whole life-time. The result of an execution must also be reported to the same “home” server. The steps of a workflow execution, or sub-workflows can be executed in a distributed manner (maybe even parallel). As different workflows can be started at different servers, this approach is only partially centralized, namely, with respect to individual workflows. A special advantage is beside the ease of implementation, the ease of monitoring (a very important practical aspect).

2.2.2 Migration. In this approach the workflow instance carries its entire state from one site to the other. Therefore, a centralized control is not necessary any more. On the other side, the data set that must be transmitted over the network might be pretty large (documents, process history). This can be again reduced by a pre-distribution of build-time data (as done in MENTOR, Exotica/FMQM and METEOR2). An alternative solution is to replicate the invariant part of the data. Migration has the obvious advantage that code and data are always located on the same site, and that even if the home server exits, already migrated workflows can continue.

The real disadvantage of this approach is the difficulty of monitoring, as for this we need an instance having exact knowledge about the history of a workflow. This can be achieved if the migrating units keep informed their home servers about their progress.

3 Related work

A lot of research effort has been done to investigate the techniques of partitioning of workflow systems into components and of the distribution of such components among the participants. Much less efforts have been done into the investigation of the distributed execution of workflows, which is the main subject of this paper.

In the WfMS MENTOR (Middleware for Enterprise-Wide Workflow Management, a product of a cooperation between the University of Saarland, the ETH Zürich, and the UBS [10]), workflows can be specified with the help of state- and activity-charts. These specifications are converted by a compiler into a semantically equivalent set of components [12, 8], which are automatically distributed over the workflow servers.

The disadvantage of the approach is its rigidity; it cannot adapt to dynamic changes of the environment (e.g. a new server arises).

A similar approach with similar rigidity is followed by Exotica/FMQM (IBM Research Center Almaden [2, 7]) and by METEOR2 (Large Scale Distributed Information Systems Laboratories at the University of Georgia, USA, [3]).

In the following section a different approach is suggested, providing higher flexibility at run-time.

4 Distributed workflows in AltaVista Works

AltaVista Works (AVW) is a product of Compaq Austria, developed in cooperation with Groiss Informatics. It is based on the prototype Panta Rhei developed at the University of Klagenfurt [4].

AVW is a web-based system, all user interactions are done via a web-browser making the system platform independent on the client. On the server side the system is implemented entirely in JAVA, using the Java Database Connection (JDBC) for accessing a relational database.

For the communication between AVW-servers we use the middleware product Voyager from Objectspace, Inc. It provides simple, yet powerful mechanisms for distributing Java objects [9].

As in most other workflow management systems the workflows are represented as graphs, where the nodes are the workflow steps. Each step can be an elementary activity, a control step (if, begin of parallel execution, etc.) or a workflow itself. The elementary activities specify what is done in this step (either a manual or automatic task) and the agent of the activity. Agents can be named users or roles (an abstraction for a group of users, for example secretary or clerk). If an activity is assigned to a role, every user belonging to this role receives the task, one of them must take it for processing it.

4.1 Management of build-time data

As stated in 2.1, for proper and efficient task assignment, a global view on build-time data is necessary. In AltaVista Works data replication is used for this. There is a central master server and an arbitrary number of slave servers. The slave servers propagate any change to the master, and can poll on demand the master for the actual build-time data. The master is passive, it does not even need to know the slaves. With the help of the actual copies the slaves can make role assignment decisions autonomously. It is a disadvantage to a certain extent that the master is a central com-
ponent. Note, however, that normal workflow processing is not influenced by an unavailability of the master. Moreover, with the idea of a master server, it is easy to implement a lock manager, which we need to keep changes atomic. To reduce the effect of a master crash further, we introduce two modes of operation for changing build-time data:

- **Synchronous Change.** A change is not allowed until the master can be reached.
- **Asynchronous Change.** If the master cannot be connected, the slave makes the changes locally, and enters them into a persistent queue. Eventually the persistent queue will be emptied and the change transmitted to the master. If this would cause inconsistency at the master, the slave has to undo the previous local changes.

### 4.1.1 Management of inconsistency

Replication causes necessarily a certain danger for inconsistency. In the actual case this means that a slave server may make a role assignment decision on the basis of not entirely up-to-date build-time data. As build-time data do not change too often, this is supposed to happen rarely. Moreover, a certain part of the data are handled in AltaVista Works in a write-once manner anyway, i.e. at every modifying operation a new version is created. Thus, for these data inconsistency is not an issue. For the other kind of data we have to investigate following cases:

1. **No server can be found to execute the next step.** The slave server does not know the execution unit that is assigned to the next step. In this case, a corresponding message is sent to the system administrator.

2. **Not all potential execution units are considered.** This should not cause any problem, in the case of several candidates the assignment is arbitrary anyway.

3. **A local step is sent to a remote server.** Due to inconsistent data the slave believes that it has to send a step to another server, although the step should be processed by itself. In this case the remote server rejects the task. The slave may update its state and learn that the step is to be executed by itself. As it still has the workflow instance, it can do that immediately.

4. **The slave tries to execute a remote step locally.** The local slave mistakenly tries to execute a step, which should be passed to a remote server. It does not find, however, any human user that could execute the external part of the step (note that a workflow step consists typically of an interaction between a computer and a human). Such a situation can be detected and resolved by the system administrator with the help of the monitoring functionality.

### 4.1.2 Distribution of modifications to the master

The process of an update is presented in Fig. 1 on the example of insertion. If the user calls the *insert* method of a distributed object (an instance of any subclass of the class *DistributedObject*), this sends a message to a *Log*-object. This logs all operations that cause a change (*insert, delete and update*), and sends a *hasChanged*-message to the so-called *replication slave*.

Build-time data may have a local or a global scope. Global data are visible overall in the system. Local data can be accessed only by the users of a given local slave, they
may, however, be referenced by global data. Therefore, the replication server collects beside all modified global data also all local data referenced by the previous ones (via the getDependentObjects method). It sends the collected data to the master server (except it is the master server itself), which makes them persistent in the attached database.

If the server is not available and the system is in the asynchronous modus (see chapter 4.1), the collected data are inserted into a persistent queue for later retrieval.

4.1.3 Retrieval of modifications from the master. Slave servers has to poll periodically (or on an explicit request of the system administrator) the master for modifications. The requesting slave sends the identifier of its last valid log-entry. With that, the master can retrieve from its log file all modifications not yet seen by the requestor. It sends the modified data to the slave, which updates its database.

4.2 Distributed execution of workflows

The DWfMS is responsible for finding the right server of the execution of the steps of a workflow. Every user in AVW is assigned to exactly one server. Therefore, all activities for a specific user must be sent to his server. If a task is assigned to a role, the workflow must be sent to all servers where users with this role reside.

As discussed in 2.2, there are basically two ways for distributed workflow execution: remote execution and migration. An early version of AltaVista Works took the first approach [5]. However, experience showed that this scales poorly. As opposed to build data, the number and size of data to be considered grows rapidly with growing degree of distribution. Moreover, home server (the one which started the actual workflow) and execution server (the one executing the current step) have a too high need for communication.

For the above reasons, migration was chosen for the new version. A workflow process migrates always to that server, where the step can be executed (including the interaction with the human user).

The two main operations in controlling a workflow execution are starting and finishing an activity.

4.2.1 Step completion. If the processing of a workflow step has been completed, the finish method of a StepInstance object is called (see Fig. 2). A StepInstance represents the instance of a step and provides the necessary methods for a successful step-by-step execution of a workflow instance. As a next action the sendFinish method of a ProcessMigrator-object is called. This checks whether execution server and home server are the same. If not, it sends the changes to the home server (via receiveProcessData), mainly for monitoring purposes. This assures that the home server always knows the actual state of its workflows.

If the workflow step just finished has a successor, the workflow engine starts this step (via the start method). The isLocal method is called and either suggests not to distribute, or it returns a list of servers. In the latter case, the ProcessMigrator broadcasts a corresponding request to the server(s) (via distribute). There are two cases:

- The step is assigned to a user: In this case the entire workflow is migrated to the destination server.
- The agent is a role: In this case the task is sent to all servers where a user with the specified role exists.
When one of them takes the task, the workflow data is sent to "his" server and is removed from the other servers.

4.2.2 Parallel paths. AltaVista Works supports the parallel execution of activities of a workflow. In the case of and-parallelism all parallel paths must end before the next step can be started, in the case of or-parallelism at least one of them. If the execution reaches a starting point of parallel execution, the individual paths are regarded as independent workflow steps, i.e. independent distributions are made. The execution of a path ends if a join node is reached in the workflow. The difficulty is to decide the location of joining the parallel paths. The individual copies do not know the location of each other. The solution is that at join, all copies return to their home server, which has always up-to-date information about the workflow.

5 Conclusion and future work

The mechanisms for adding distribution to AltaVista Works has been presented. The system-wide global view on build-time data was achieved by replication of these data on all participating workflow servers. Replication enables an efficient role assignment, based on local decisions of slave servers. Using replication makes a partitioning in advance unnecessary.

To realize distributed workflow execution, migration has been chosen. As opposed to build-time data, in this case it is necessary to avoid a central component. Although any server that starts a workflow remains its home server for the entire execution, such a home server is "central" only for its own workflows. This reduces the probability of getting any server to a bottleneck. If a home server exits, only the workflows initiated by this particular server are affected, all other workflows may continue.

The distributed version of AVW is currently used by two customers, one of them having installed 7 servers with several hundred users.

For the future, the following extensions are planned:

1. We want to open the replication mechanism for application programmers. Some data in distributed workflows are not process specific but used by all processes of a specific application, for example address lists or other application specific master data. Using the replication mechanism, application programmers can distribute arbitrary data over the network of AVW servers.

2. A tool for monitoring the state of replication on all servers in the network as well as the states of the migration queues on the servers is currently in development. In networks with dozens of servers widely distributed, such a tool is necessary to efficiently maintaining the system.

References


