Efficient Distributed Workflow Management Based on Variable Server Assignments

Thomas Bauer and Peter Dadam

University of Ulm, Dept. of Databases and Information Systems
{bauer,dadam}@informatik.uni-ulm.de
http://www.informatik.uni-ulm.de/dbis

Abstract. For enterprise-wide and cross-enterprise workflow (WF) applications, the load of the WF servers and the amount of communication in the subnets may become a bottleneck. This paper shows how a distributed WF control can be realized in a way that the load of the components at run time is minimized. For that purpose, the control of a WF instance may migrate from one WF server to another. The WF servers are assigned to the WF activities in a way that minimizes the communication load. The server assignments are determined at build time by analyzing the WF model with respect to the actor assignments. As these actor assignments may depend on preceding activities, static server assignments are not always reasonable. Hence, so-called variable server assignment expressions are introduced, which allow dynamic server assignment without expensive run time analyses.

1 Introduction

Workflow Management Systems (WfMSs) offer a promising technology for the implementation of process-oriented information and application systems. A significant limitation of current WfMSs, however, is their insufficient scalability, which is caused by the use of one central WF server. Already the processing of a single WF activity may require the transfer of multiple messages between the WF server and its clients; e.g., to transmit input and output data of the activity, to update worklists, or to invoke activity programs. As soon as the number of users increases, e.g., when the WF-based application may be accessed via the Internet, such a central WF server may become overloaded. For this reason, several multi-server approaches have been proposed in WF literature [1, 9, 16, 21].

1.1 Motivation

Having a closer look at enterprise-wide and cross-enterprise WF applications, it becomes obvious that not only a central WF server, but also the corresponding subnet may become a bottleneck. As an example consider a (simplified) loan request WF as shown in Fig. 1. Let us assume that the input and output data of the activities 1, 3, 4,

B. Wangler, L. Bergman (Eds.): CAiSE 2000, LNCS 1789, pp. 94-109, 2000

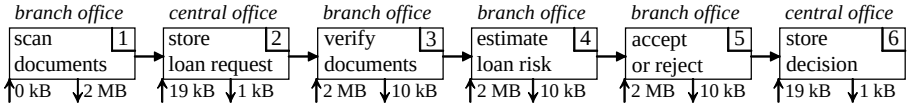

Fig. 1. Loan request WF (↑/↓: average size of the input / output parameter data of the activity)

and 5 contain scanned documents with a total average size of 2 MB[1]. Let us further assume that a bank has 30 branch offices, which are connected with the central office by a wide area network (WAN). Each branch office shall have 10 clerks; i.e., there are 300 clerks in the branch offices altogether. If we assume that a clerk requires 5 min = 300 sec for the execution of one activity instance, then one activity per second is executed on the average, each of them requiring 2 MB of parameter data. Hence, a central WF server has to transmit 2 MB/sec = 16 Mbit/sec of parameter data. Obviously, this would overload an Ethernet-based local area network (LAN). In order to get further insights, a simulation (c.f. section 5) of the scenario described was performed. It leads to the following results: The load[2] of the subnet of the central WF server is 15.7 Mbit/sec. In this context, it is very attractive to control each WF by the WF server of the corresponding branch office. In this case, no subnet has a load higher than 0.5 Mbit/sec. Another very important aspect in distributed systems (especially in widely distributed systems) is the data volume transferred by the gateways. If a central WF server is used in the sketched scenario, the gateways have to transmit 15.6 Mbit/sec. This corresponds to 243.8 permanently used ISDN channels (each with a capacity of 64 kbit/sec). If distributed WF control is used, instead, the gateways have only to transmit 80.6 kbit/sec; i.e., only 1.3 ISDN channels are required! This example shows, in a very impressive way, that an appropriate distributed WF control is not only important to avoid an overloading of the WF servers, but it is also essential to avoid bottlenecks in the communication network.

1.2 Distributed WF Execution in the ADEPT-WfMS

Before we explain the basic ideas of this paper, we shortly summarize our previous work on distributed WfMSs. Performance and scalability, especially in the context of enterprise-wide and cross-enterprise WF applications, belong to the central research issues of the ADEPT research project[3] [19]. An important goal of the ADEPT approach is the reduction of the total communication load caused by the WfMS. In order to achieve this, a WF instance may be controlled "piecewise" by different WF servers (see Fig. 2). Consequently, the control of a particular WF instance migrates from one WF server to another, if this helps to avoid communication across subnet boundaries

[1] Even much larger data volumes are possible in such enterprise-wide application scenarios; e.g., if multimedia data have to be transferred between different activities.

[2] This load is lower than 16 Mbit/sec because it is not possible to use the whole capacity of the clerks since this would cause their worklists to overflow. Therefore, a clerk of a branch office performs less than 1 activity/sec.

[3] ADEPT stands for Application Development Based on Encapsulated Pre-Modeled Process Templates.

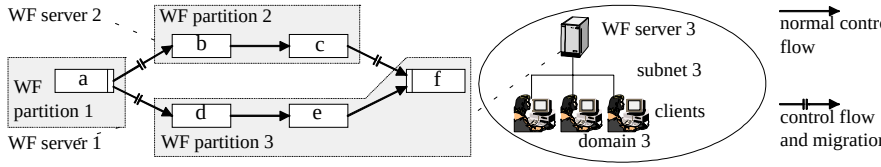

Fig. 2. Partitioning of a WF graph and distributed WF execution

when executing WF activities [3]. The concrete server assignments of the activities are completely computed at build time. They are based on probability distributions (PD) of activity executions within the individual subnets (which, in turn, are derived from the distribution of users and their roles; see [3] for details). That is, before creating a WF instance, the WF servers of the partitions are completely predetermined. Such static server assignments for activities show good results, if actor assignments (*ActorAss*) are of the kind "role = physician" or "role = nurse ∧ department = surgery". In these cases, the set of possible actors performing an activity (as precondition for the calculation of the PDs) can be determined already at build time.[4]

1.3 Problem Statement and Contribution of the Paper

Unfortunately, in practice, there are many cases in which concrete actor assignments depend on preceding activities (called “dependent actor assignments” for short). For example, it may be required that the same physician who examines a patient (activity *x*) must also write the corresponding medical report (subsequent activity *k'* in Fig. 3). As another example take a patient who must be cared by a nurse from the same department (activity *k*) in which he or she was previously examined. In the context of such dependent actor assignments, the following important problem occurs: The organizational unit (in the following called unit for short) to which the actors performing the activities *k* and *k'* belong, is not fixed until activity *x* is executed. What does this mean for the assignment of appropriate WF servers to the activities? If dependent actor assignments occur, it would be possible to calculate the best static server assignments by using conditional probabilities. Since static server assignments can not respect the dependencies, in too many cases the dependent activities would be controlled by the “wrong” WF server.

The main contribution of this paper is to develop a multi-server WfMS, which is based on the concept of *variable server assignments*. Instead of using only static server assignments, like "activity *x* is controlled by server S_5", in this approach logical server assignment expressions can be used as well. Such an expression may determine, for example, that "activity *k* is controlled by the server, which is located in the subnet of the actor performing the preceding activity *x*". These expressions allow to take run time data of the WF instance into account, when determining the concrete WF server controlling the activity instance. And, best of all, they create almost no additional overhead at run time when compared with the static case. In this paper, we

[4] The calculation relies on the assumption that no massive changes take place between build time and the execution of instances of this WF type.

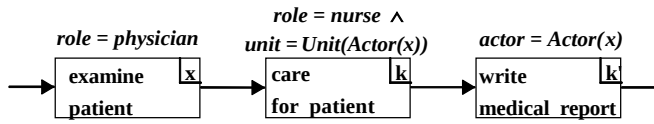

Fig. 3. Examples for dependent actor assignments

present the cost model and the distribution algorithm used to calculate appropriate variable server assignment expressions at build time. In addition, we present some simulation results which show that the communication load can be reduced significantly by using variable server assignments. Despite it is commonly accepted that distributed WF execution is essential – to our knowledge – $ADEPT_{distribution}$ is the only approach that treats the problem of dependent actor assignments in a distributed WfMS.

In the next section the conditions for the applicability of our approach are summarized. Section 3 describes possible solutions of the problem. The nucleus of this paper is presented in Section 4. It describes the algorithms for determining the server assignment expressions at build time. The efficiency of our approach is shown in Section 5 by means of simulations. In Section 6 related approaches are discussed. The paper concludes with a summary and an outlook in Section 7.

2 Assumptions and Preconditions

The approach introduced in this paper describes how load distribution is realized by the use of variable server assignments in $ADEPT_{distribution}$, the distributed variant of the ADEPT-WfMS. It is only based on the following realistic assumptions:

1. The WfMS uses an organizational model in which actors can be associated with organizational units as well as with roles.
2. The actors are assigned to an activity at run time by means of a selection predicate like "role = physician ∧ unit = radiology". This predicate may also reference preceding activities (dependent actor assignments, cf. Fig. 3). Otherwise, it is assumed that the actor of an activity is determined independently from other activities.
3. The WfMS uses several subnets as well as several WF servers. To simplify the following discussion, it is assumed that each subnet is equipped with one (and only one) WF server. Each of these WF servers may control each WF type and its partitions, respectively. A user may be permanently assigned to one or several subnets.
4. Each WF server can serve all WF clients registered in the WfMS, not only those of its domain (a subnet together with the corresponding WF server and clients is called *domain*).
5. The actors who may potentially execute a certain activity are not necessarily located in the same subnet.
6. The number of persons within a subnet who qualify to perform a certain activity does not change significantly between build time and run time.
7. The topology of the communication network does also not change significantly.

3 Possible Solution Approaches

As already mentioned, our aim is to identify server assignments which minimize the total communication costs. Special attention must be paid to dependent actor assignments since they occur very often in practice. In principle, there are several possible approaches for the assignment of WF servers to activities: The simplest and most frequently applied method is to assign a fixed server (statically) to each activity at build time or at the time the WF instance is started. This solution is not satisfactory for WFs for which dependent actor assignments have to be supported as well (see Section 1.3). Obviously, the best server assignments can be achieved if the WF server for the activity to be executed is determined after the preceding activity has finished. At this point in time, complete and current run time data of the WF instance is available. Thus the most suitable WF server can be determined. Unfortunately, this solution is not applicable in practice due to its high costs, in general. The execution of the necessary analyses would heavily burden the WF servers and thus negatively affect the performance of the WfMS. This is not acceptable for production WfMSs that have to cope with a high load.

Therefore, $ADEPT_{distribution}$ follows a strategy which combines a static pre-calculation (at build time) with dynamic aspects (i.e., evaluation of run time data) and, by doing so, combining the advantages of both approaches. Instead of a static WF server assignment, logical expressions for server assignments are created at build time, if this is reasonable. They reference run time data of the WF instance, thus making it possible to simply and efficiently determine the appropriate WF server at run time.

The main challenge of this approach is how to get suitable variable server assignments for the activities. One possibility would be to let the WF designer specify them explicitly at build time (analogous to the actor assignments). Unfortunately, the WF designer, without further support or information, will hardly be in the position to estimate the load in the subnets for a particular distribution. Therefore, it is essential to support him actively by means of a sophisticated WfMS modeling tool, which estimates the load of each system component (server, subnet, gateway) for the server assignments taken under consideration. This, in turn, requires an appropriate cost model. This cost model and the calculation of appropriate server assignments are described in the next section.

4 Determination of Optimal Server Assignments

In this section we describe how appropriate server assignments for WF activities can be calculated already at build time.

4.1 Cost Model

A cost model for determining the load of the system components which are critical under performance aspects (servers, subnets, gateways), has to consider at least the following costs:

- Costs for the transfer of parameter data (between a WF server and a WF client) when starting and finishing activity programs
- Costs for refreshing worklists at the client site
- Costs for data transfer between activity programs and external data sources
- Costs for migrating the control of a WF instance to another WF server (migration costs for short)

In addition to the information available from the WF template and from the organizational model, the estimated execution frequency and the average amount of data to be communicated (e.g., the size of the parameter data) have to be determined for each activity of the WF template. For already released WF templates, these data may be obtained by analyzing the audit trails of executed WF instances. Otherwise, they have to be estimated.

In the sequel we develop formulas which describe the data volume to be transported for a single system action (e.g., "start activity" or "migrate WF") and an individual system component. Subsequently, these simple formulas are used to construct a comprehensive formula, which describe the total load of the components. Thereby, $ExProb_k(i, j)$ denotes the probability that activity k is controlled by the WF server in subnet i and is processed by a user in subnet j (Table 1 summarizes the expressions which are used in this chapter). $MigrProb_{k,l}(i, j)$ denotes the probability that, when moving from activity k to activity l, the control migrates from the WF server i to the server j. These probabilities depend on the server assignments selected. We will show in Section 4.4 and 4.5 how they can be determined. For the moment, we consider them as given.

The expected value for the data volume (abbreviated by *data volume* in the following) for the *execution of an activity* (transport of input and output parameter data between WF server and client) is calculated as follows: The data volume emerging at server i for the execution of activity k results as the probability that server i controls activity k, multiplicated by the average amount of data to be transported:

$$Vol^{Act}_{Server,i}(k) = \left(\sum_j ExProb_k(i, j) \right) \cdot (in\_parameter\_size_k + out\_parameter\_size_k) \qquad \textbf{(1)}$$

Table 1. Abbreviations used in this paper

Name	Meaning
$ActorAss_k$	actor assignment of activity k
$ActorProb_k(D\|S)$	portion of the actors in the domain (subnet) D, given that activity k is controlled by server S
$DepMigrProb_{x,k}(j\|i)$	probability of a migration to the WF server in subnet j (when moving from activity x to activity k), given that activity x was controlled by the server in subnet i
$DepServProb_k(S\|u)$	probability that activity k is controlled by server S, given that actor u performs this activity
$ExProb_k(i,j)$	probability that activity k is controlled by the WF server of subnet i and processed by an actor of subnet j
$MigrProb_{x,k}(i,j)$	probability of a migration from the WF server in subnet i to the WF server in subnet j (when moving from activity x to activity k)
$ServAss_k$	server assignment of activity k
$ServProb_k(S)$	probability that activity k is controlled by the WF server S (in subnet S)

The *load of the subnets* is estimated as follows: In subnet *i* communication with respect to the execution of activity *k* takes place, either when the server is located in *i* or when a user of subnet *i* executes the activity. If both is valid, the communication must be counted only once (therefore $j \neq i$):

$$Vol^{Act}_{Subnet,i}(k) = \left(\sum_{j} ExProb_k(i, j) + \sum_{j \neq i} ExProb_k(j, i) \right) \cdot (in\_parameter\_size_k + out\_parameter\_size_k) \quad (2)$$

The expected *data volume at the gateway* from subnet *i* to *j* ($i \neq j$) is calculated as follows:

$$Vol^{Act}_{GW,i,j}(k) = ExProb_k(i, j) \cdot in\_parameter\_size_k + ExProb_k(j, i) \cdot out\_parameter\_size_k \quad (3)$$

The expected data volumes for refreshing the worklists ($Vol^{WL}(k)$) as well as for the communication of activity programs with external data sources ($Vol^{Ext}(k)$) can be calculated in a similar way (a detailed description can be found in [7]). The latter cost factor can be used to calculate a suitable allocation of externally stored application data (which is only referenced by the WF instances), in case this distribution is not predetermined already.

The migration costs at the transition from activity *k* to activity *l* are, of course, of special interest in the context of this paper. Incoming as well as outgoing migrations have to be considered. To calculate the expected data volume for server *i*, the average data volume communicated for this migration is multiplied by the probability that server *i* is involved in the migration. Here, it has to be noted that no communication takes place if the servers of the activities *k* and *l* are identical (therefore $j \neq i$).

$$Vol^{Migr}_{Server,i}(k, l) = \sum_{j \neq i} \left(MigrProb_{k,l}(i, j) + MigrProb_{k,l}(j, i) \right) \cdot migration\_size_{k,l} \quad (4)$$

The data volume caused by the migration in the subnets affected is the same as for the WF servers:

$$Vol^{Migr}_{Subnet,i}(k, l) = Vol^{Migr}_{Server,i}(k, l) \quad (5)$$

The gateways have to transport the following amounts of data:

$$Vol^{Migr}_{GW,i,j}(k, l) = MigrProb_{k,l}(i, j) \cdot migration\_size_{k,l} \quad (6)$$

For each system component, these amounts of data have to be multiplied by the execution frequencies *ExFreq(k)* of the activities (*k*) or of the migrations *MigrFreq(k, l)*, respectively. Then they have to be added up for all activities of all WF types (*WFTypes*) in order to determine the corresponding load of this component. For the servers the load is calculated as follows ($Load_{Subnet,i}$ and $Load_{GW,i,j}$ are determined analogously):

$$Load_{Server,i} = \sum_{wf \in WFTypes} \sum_{k \in wf} ExFreq(k) \cdot Vol^{Act}_{Server,i}(k) + ExFreq(k) \cdot Vol^{WL}_{Server,i}(k) + ExFreq(k) \cdot Vol^{Ext}_{Server,i}(k) + \sum_{l \in wf \wedge l \neq k} MigrFreq(k, l) \cdot Vol^{Migr}_{Server,i}(k, l) \quad (7)$$

These loads are weighted with the component specific cost factors $C_{Server,i}$, $C_{Subnet,i}$, and $C_{GW,i,j}$ and they are added up in order to get the total costs. These cost factors specify the costs for transferring one byte over the server, the subnet, and the gateway. Thus, the WF designer can influence the load of each component. A high value of $C_{GW,i,j}$ has the effect that, for example, the WAN-connection (gateway) is used little. Hence, the target function to be minimized is as follows:

$$T = \sum_i C_{Server,i} \cdot Load_{Server,i} + \sum_i C_{Subnet,i} \cdot Load_{Subnet,i} + \sum_i \sum_{j \neq i} C_{GW,i,j} \cdot Load_{GW,i,j} \quad \textbf{(8)}$$

The following section describes which server assignment expressions are possible for the activities. Section 4.3 shows how the server assignments can be selected in a way that minimizes *T*.

4.2 Server Assignment Expressions

For static server assignments, the identifiers of the WF servers are directly used as expressions. For variable server assignments, however, expressions may reference run time data of the WF instance. For most of the practically relevant scenarios, the location of the WF server or the actor of a preceding activity is referenced. In special cases the usage of additional WF control data (e.g., the start time of an activity), the inclusion of WfMS-external data (e.g., parameter data of activities), or the evaluation of mathematical functions or logical expressions may be reasonable as well. While server assignments of the first kind (see 1 - 4 below) can automatically be deduced from the WF model, the assignment expressions for the special cases (5.) must be explicitly specified by the WF designer. The server assignment expressions which are supported by $\text{ADEPT}_{\text{distribution}}$ are as follows:

1. $ServAss_k = S_i$
   Server S_i is statically assigned to activity *k*.
2. $ServAss_k = Server(x)$
   Activity *k* shall be controlled by the same server as activity *x*.
3. $ServAss_k = Domain(Actor(x))$
   Activity *k* shall be assigned to the server that is located in the domain of the user who has executed activity *x*.
4. $ServAss_k = f(Server(x))$ or $\text{ServAss}_k = f(Domain(Actor(x)))$
   A function *f* can be applied to all server assignments of type 2 and 3. Assume, e.g., that activity *k* is assigned to the manager of the actor who works on activity *x*. This manager may belong to a different domain than the actor of activity *x* (e.g., if he is assigned to a different department). In such cases, a simple mapping function can be used to perform the desired transformation. In [7] we describe how a suitable function *f* can be deduced automatically.
5. $ServAss_k$ = any given expression
   The WF designer may specify own server assignment expressions, which do not correspond to any of the expressions of type 1 - 4. Since they cannot be analyzed by the WfMS, the PDs cannot be calculated automatically. Therefore, the designer has to provide this information as well. The PDs are required by the algorithms of Section 4.4 in order to perform the calculations.

4.3 Determination of Optimal Server Assignments

In order to determine an optimal distribution of the activities of a WF template, in principle, the costs of all possible server assignments for all activities have to be computed and, by doing so, the server assignments with the minimal costs can be selected. Since a WF template may easily comprise more than 100 activities and each

server assignment may reference any predecessor, this approach is not feasible, in general, due to its complexity $O(\#Act^{\#Act})$, where *#Act* denotes the number of activities of the WF template. Therefore, we suggest an algorithm (see Algorithm 1) which performs this task with polynomial run time complexity. It is based on a greedy approach and calculates the optimal result or a result that is close to this optimum for the practically relevant cases. It consists of two phases, which work as follows:

In Phase 1 a legal initial solution is determined. This is achieved by computing the optimal server assignment for each activity under the assumption (for the moment) that migration does not cause any costs. For each activity *k* all possible server assignments $PotServAss_k$ (using all possible expressions of type 1 - 4, c.f. Section 4.2) are tested. The cost calculation is performed by the function *calculate_costs()*, which uses the cost model described in Section 4.1.

As only single activities are analyzed in Phase 1, the result vector *ServAss* may contain unprofitable migrations as well. In Phase 2, Algorithm 1 examines which of these migrations are really reasonable. In order to eliminate undesirable migrations, the partitions *P* are inspected (all activities of a partition are controlled by the same WF server). It is analyzed, whether it is advantageous to combine such a partition *P* with a direct predecessor partition or a direct successor partition, in order to eliminate the migration between them. The motivation for this analysis is that it may be not worth migrating the (complete) WF instance from one server to another and back for only a few activities. Firstly, the algorithm considers small partitions *P*. By combining them with adjacent partitions they, step by step, form larger groups of activities among which no migrations take place. This integration process is continued until there are no more unprofitable migrations.

Algorithm 1: Calculation of the Server Assignments for a WF Type (Process Template)

Phase 1:
for each activity $k \in$ *ProcessTemplate* (in partial order corresponding to the control flow) **do**
  $MinCost = \infty$;
  for each $ServAss_k \in PotServAss_k$ **do**
    Cost = calculate_costs(ServAss, k); // costs only of the activity *k*
    if *Cost < MinCost* **then** $OptServAss = ServAss_k$; *MinCost = Cost*;
  $ServAss_k = OptServAss$;
Phase 2:
MinCost = calculate_costs(ServAss, all); // costs of the whole WF (incl. migrations)
for *PartSize* = 1 **to** *#activities(ProcessTemplate)* **do**
  for each *P*: $|P|=PartSize$, *P* is a maximal subgraph with $\forall l_1,l_2 \in P$: $Server(l_1) = Server(l_2)$ **do**
    OptServAss = NULL;
    for each $a \notin P$: $\exists\, l \in P$: $a = predecessor\ (l) \vee a = successor\ (l)$ **do**
      for each $l \notin P$ **do** $TestServAss_l = ServAss_l$;
      for each $l \in P$ **do** $TestServAss_l = ServAss_a$;
      TestCost = calculate_costs(TestServAss, all);
      if *TestCost < MinCost* **then** $OptServAss = ServAss_a$; *MinCost = TestCost*;
    if $OptServAss \neq NULL$ **then**
      for each $l \in P$ **do** $ServAss_l = OptServAss$;

4.4 Calculation of the Probability Distributions

After having explained which server assignments are suitable for an activity and how the corresponding costs can be calculated, the question remains, how the PDs $ExProb_k(i,j)$ and $MigrProb_{x,k}(i,j)$ can be determined.

As server assignments depend on run time data of the WF instance, different instances of a particular WF activity may be controlled by different WF servers. The probability that activity k is controlled by server S is denominated with $\boldsymbol{ServProb_k(S)}$. As different instances of k can be located in different units (and therefore may be controlled by different servers), the actor PD of k may be different for each server S. The portion of the actors of activity k that are located in domain D is called $\boldsymbol{ActorProb_k(D|S)}$, for the case that server S controls activity k. In the sequel we describe how $ServProb_k(S)$ and $ActorProb_k(D|S)$ can be determined. Once they are known, the probability that activity k is controlled by server S and that is executed by a user in domain D is given by: $ExProb_k(S, D) = ServProb_k(S) \cdot ActorProb_k(D|S)$.

4.4.1 Calculation of the Server Probability Distribution $ServProb_k(S)$

$ServProb_k(S)$ denotes the probability that server S controls activity k. It results from the server assignment $ServAss_k$ and the PDs of the activity x, that is referenced in $ServAss_k$. In Algorithm 2, the activities of the WF template are analyzed in the (partial) order defined by the control flow. Since activity x must be a predecessor of k, these PDs are, therefore, already known when analyzing activity k. In the following we describe how the server PD $ServProb_k(S)$ is calculated, if the server assignments as defined in Section 4.2 are considered. Case 5 is not considered here (and further on), as for this server assignment the PD has to be provided by the WF designer.

Algorithm 2: Calculation of the Server Probability Distribution $ServProb_k(S)$

case $ServAss_k = S_i$:	$ServProb_k(S) = \delta_{S,S_i}$ [5]	(1)
case $ServAss_k = Server(x)$:	$ServProb_k(S) = ServProb_x(S)$	(2)
case $ServAss_k = Domain(Actor(x))$:	$ServProb_k(S) = ActorProb_x(S)$ with $ActorProb_x(D) = \Sigma_S ActorProb_x(D\|S) \cdot ServProb_x(S)$	(3)
case $ServAss_k = f(Server(x))$:	$ServProb_k(S) = f(ServProb_x(S))$	(4a)
case $ServAss_k = f(Domain(Actor(x)))$:	$ServProb_k(S) = f(ActorProb_x(S))$ with $ActorProb_x(D) = \Sigma_S ActorProb_x(D\|S) \cdot ServProb_x(S)$	(4b)

Explanations:

(1) Since activity k is always controlled by the server S_i, it follows $ServProb_k(S) = 1$ for $S = S_i$ and $ServProb_k(S) = 0$ otherwise.

(2) The same server is used for activity k as for activity x. Therefore the server PDs are identical.[6]

[5] Kronecker symbol: $\delta_{i,j} = 1$, if $i = j$ and $\delta_{i,j} = 0$, if $i \neq j$.

[6] Assuming that there are no branches between the activities x and k, which depend on the choice of the server of activity x. Such facts are difficult to detect because they concern data elements. Therefore, for branch and loop conditions, we have generally assumed that their result is independent of the current server.

(3) The server of activity *k* is located in the domain of the actor of activity *x*. Therefore the server PD of activity *k* results from the actor PD of activity *x*. It does not matter which server has controlled activity *x*. Hence a server independent actor PD $ActorProb_x(D)$ is created by the weighted sum of the actor PDs for the different servers.
(4a) $ServProb_k(S)$ is calculated as described for case 2. Afterwards the function *f* is applied to the result (cf. [7] for details).
(4b) The same as case 3, but in addition *f* is applied to the result.

4.4.2 Calculation of the Actor Probability Distribution $ActorProb_k(D|S)$

$ActorProb_k(D|S)$ describes the probability that the actor of activity *k* is located in domain *D*, if the activity is controlled by server *S*. As an example take a hospital with 3 wards, each of which owning its own server. Assuming that only nurses of ward 1 (domain 1) are in charge of the patients of ward 1, it is reasonable that server 1 controls this activity. Hence, an actor PD $ActorProb_k(D|1) = (1,0,0)$ results. Analogously, for server 2 we obtain $ActorProb_k(D|2) = (0,1,0)$ and for server 3 $ActorProb_k(D|3) = (0,0,1)$.

In the following, we present Algorithm 3, which calculates the actor PD $ActorProb_k(D|S)$. It considers all possible actors of activity *k* and determines for each of them the corresponding domain *D* (from the organizational model). Furthermore, it determines the server *S* that controls the activity *k* if it is performed by user *u*. The user is reflected in the actor PD $ActorProb_k(D|S)$ for the corresponding server *S* and the domain *D*. Thus, the calculation of $ActorProb_k(D|S)$ is performed according to the definition of the actor PD (see Section 4.4). Different servers may qualify – each with a certain probability – for the control of activity *k* if it is performed by user *u*. These probabilities are calculated by the function *DepServ(k, "Actor=u")* (see below) and are stored in the vector $DepServProb_k(S|u)$. $DepServProb_k(S|u)$ determines the weight, with which the user *u* is reflected in $ActorProb_k(D|S)$.

Algorithm 3: Calculation of the Actor Probability Distribution $ActorProb_k(D|S)$
Actors = {*u* | user *u* qualifies as actor of activity *k*};
for each $u \in Actors$ **do**
  $D = Domain(u)$;
  $DepServProb_k(S|u) = DepServ(k, "Actor = u")$;
  for each *S* **do** $ActorProb_k(D|S) = ActorProb_k(D|S) + DepServProb_k(S|u)$;
normalize each line of $ActorProb_k(D|S)$ such that $\forall S: \Sigma_D ActorProb_k(D|S) = 1$;

The calculation of the *server PD* $DepServProb_k(S|u)$ for a certain user *u* is performed similar to the calculation of the user-independent server PD $ServProb_k(S)$ (c.f. Section 4.4.1). $DepServProb_k(S|u)$ does not only depend on the server assignment of activity *k*, however, but also on its actor assignment. In the following, due to lack of space, only some selected examples are presented. A comprehensive discussion of all possible cases of server and actor assignments can be found in [7].

- If the server assignment is static ($ServAss_k = S_i$) the calculation is trivial as the server is always the same: $DepServProb_k(S|u) = \delta_{S,S_i}$.
- If the actor assignment is independent from other activities (e.g., "*role = physician*") the server PD is independent from user *u*. Thus, the user-independent server PD can be adopted: $DepServProb_k(S|u) = ServProb_k(S)$.

- In Fig. 4a, the same physician, who works on activity *x* also performs activity *k*. Assume that $DepServProb_k(S|u)$ for user *u* = "Dr. Smith" from domain 3 shall be calculated. Because of $ActorAss_k = Actor(x)$, Dr. Smith must have performed activity *x* as well. Since the server of activity *k* is allocated in the domain of the actor of activity *x*, it is allocated in the domain of Dr. Smith. As this is domain 3, the result is $DepServProb_k(S|u) = (0,0,1)$.
- In Fig. 4b, activity *k* is performed by another actor than activity *x*, but the actors belong to the same unit. Assume that $DepServProb_k(S|u)$ for the nurse Jane from the unit ward 2 shall be calculated. In this case, all users with the role physician who belong to the unit ward 2 have to be considered. These are exactly those users who could have executed activity *x* if activity *k* is performed by Jane. For each of these physicians the domain *D* is calculated because it determines – if this physician has performed activity *x* – the location of the server of activity *k*. This domain *D* is reflected in $DepServProb_k(D|u)$. Finally, $DepServProb_k(S|u)$ is normalized such that $\Sigma_S DepServProb_k(S|u) = 1$ holds.

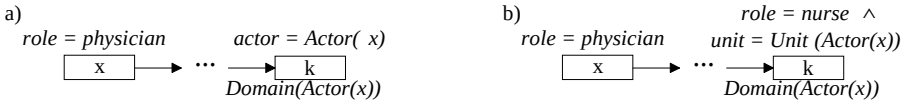

Fig. 4. Examples for the calculation of the server probability distribution $DepServProb_k(S|u)$

Additionally to these aspects, it has to be noted that there are users who work part-time, or who work only part of the working day with the WfMS, or who may work in several domains. These users must not be treated in the same way as users working "full-time" only in one domain, because they produce less load in the single domain. In $ADEPT_{distribution}$, this circumstance can be modeled by assigning a weight *Weight(u)* to each user *u*. This weight is used in Algorithm 3 to add $DepServProb_k(S|u)$ proportionally. Further possible weights are discussed in [7].

4.5 Migration Costs

In Phase 1 of Algorithm 1, the migration costs are ignored. In Phase 2, these costs are included. In order to calculate the migration costs, the migration probability $MigrProb_{x,k}(S_1,S_2)$ – with which the WF is migrated from server S_1 to server S_2 – has to be determined.

The probabilities for migrations from activity *x* to activity *k* can be described by a *migration matrix*. An entry $DepMigrProb_{x,k}(S_2|S_1)$ of this matrix describes the conditional probability that the WF migrates to server S_2, if the preceding activity *x* was controlled by server S_1 ($DepMigrProb_{x,k}(S_2|S_1)$ is defined as *P(server* S_2 *controls activity k | server* S_1 *controls activity x)*). Hence the migration probability results as: $MigrProb_{x,k}(S_1, S_2) = ServProb_k(S_1) \cdot DepMigrProb_{x,k}(S_2|S_1)$. In the following we describe how the matrix *DepMigrProb* can be determined.

By analyzing the server assignments, it can be deduced which sets of activities of a WF instance are always controlled by the same server. At the transition between two

activities *x* and *k* belonging to the same set, the WF never migrates. This results in: $\forall S_1, S_2$: $DepMigrProb_{x,k}(S_2|S_1) = \delta_{S_1,S_2}$

If the activities *x* and *k* are controlled by different WF servers, the usage of the server PD offers a simple method for the estimation of $DepMigrProb_{x,k}(S_2|S_1)$. The server PD $ServProb_k(S)$ describes the probability that the WF instance is located at the server S_2 after the migration. So the resulting migration probabilities may be approximated as: $\forall S_1, S_2$: $DepMigrProb_{x,k}(S_2|S_1) = ServProb_k(S_2)$

More sophisticated algorithms for calculating $DepMigrProb_{x,k}(S_2|S_1)$, which consider dependencies between the actors resp. the servers of the activities *x* and *k*, are presented in [7].

5 Efficiency of the ADEPT$_{distribution}$ Approach

In this section we demonstrate that the overall network load can be significantly reduced by the use of variable server assignments. For this purpose, we simulated the execution of a clinical WF when it is controlled by a central WfMS, by a distributed WfMS that does not use migrations, by a WfMS with static server assignments, and by a WfMS with variable server assignments. For further simulations and details concerning the simulated application scenarios, the simulation environment, and the interpretation of the results we refer to [5, 6].

Since the effects of variable server assignments shall be examined, the simulated WF contains dependent actor assignments. The simulation component simulates in a lifelike fashion the execution of many instances of this WF. At the same time it memorizes all occurring communications. This information is used to compute the total load of all WF servers, the average load per WF server, the load per subnet, and the load of the gateways. For each case considered, the load of the central WF server is used as reference basis. It is defined as 100. Fig. 5 shows the result of the simulation. To avoid confusion: We have been interested to show the (positive and negative) aspects of distribution and server migration with respect to server load and network load. To provide a reasonable reference basis for comparing the loads, we selected a scenario where a central WF server was not overloaded. It, therefore, could

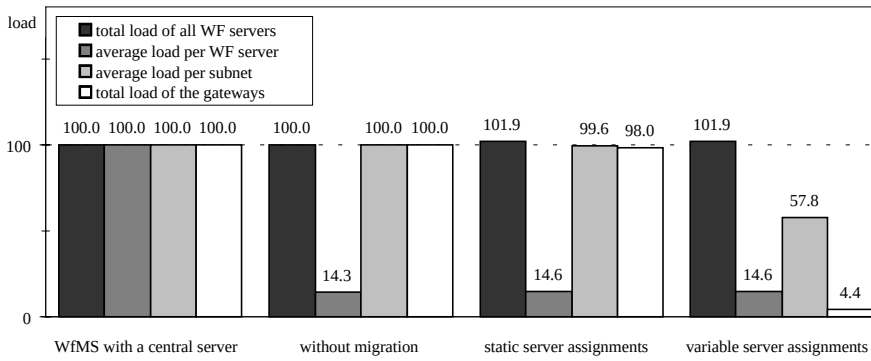

Fig. 5. Result of the simulation of a clinical WF

serve as reference point (100). This value (server load) cannot be improved in the distributed case because the synchronization overhead and the migration costs increase it. The simulation shows that variable server assignments significantly reduce the load per subnet and the data volume that has to be transferred by the gateways: The average load per subnet is 57.8% of the central case and the data volume transferred by the gateways results as only 4.4%.

6 Discussion

This section summarizes important concepts for scalable WfMSs and outlines some concrete approaches. For a more detailed discussion we refer to [4, 5, 6].

One extreme for the distribution model of a WfMS is a central server, which controls all activities. As it has to manage the whole system load, obviously, it represents a potential bottleneck. Some research prototypes which do not primarily deal with scalability issues (e.g., Panta Rhei [13], WASA [22], [10]) and most of the commercial systems belong to this category. The other extreme is a completely distributed system, which does not use any WF servers at all (e.g., Exotica/FMQM [2], INCAS [8]). In such a system, a WF migrates to the machine of the user who wants to perform an activity. With this approach, no server assignments are necessary as well.

Many approaches use multiple WF servers. As these systems require a strategy for the assignment of the servers to the activities, they are classified according to this criterion. In [1] identical replicates (clusters) of the WF engine are used. Whole WF instances are controlled by a randomly selected cluster. Therefore, no server assignments are necessary. In [20] the systems CodAlf and BPAFrame are described, which allocate an activity's WF server on the machine of the corresponding application (e.g., a DBMS). A similar approach has been suggested by METEOR$_2$ [11] and by METUFlow [12].

Like ADEPT$_{distribution}$, several approaches try to allocate the servers close to the actors of the activities. MENTOR [16] partitions state/activity charts in a way that allows to formally verify the equivalence of the distributed executions to the central case. All users of one activity have to belong to the same "processing entity". The server of this processing entity is selected for that activity. The same distribution strategy is followed by WIDE [9] using CORBA remote object access instead of migrations. In MOBILE [15] the different aspects of a WF are treated by different servers but there is no migration. This approach was extended in [21] by selecting the server for each (sub) WF at run time. The TEAM model [17] considers cooperating organizations. In this scenario, the locations of the WF servers are predetermined.

In ADEPT$_{distribution}$, the communication costs are considered by the distribution algorithm. That is, it takes into account (and tries to avoid the case) that the communication system can become a bottleneck. To our knowledge, it is the only approach that analyzes the load in order to minimize the communication costs by means of a suitable distribution. The other systems do not offer variable server assignments; the influence of dependent actor assignments on the distribution is not considered.

7 Conclusion and Outlook

WfMSs that allow the explicit modeling of the flow of control, the flow of data, and the dynamic assignment of actors to the activities offer a promising technology for the realization of enterprise-wide, process-oriented application systems. The flexibility achievable with these systems, however, is reflected in a relatively high communication load between the control components (the WF servers) and the application components (the WF clients). In WfMSs with hundreds of WF clients and thousands of active WF instances, the resulting load of the WF servers as well as of the communication network plays an essential role for the response times and also for the usability of the WfMS in total.

A possibility to reduce the network load is to keep, if possible, the communication between WF servers and their WF clients local within one subnet; i.e., to avoid subset-spanning communications. This can be achieved, if a WF instance is not necessarily completely controlled by the initiating WF server. Instead, its control may "migrate" to other WF servers, if necessary. In [3] we have presented an approach using static server assignments. In this paper, we have significantly extended this approach and examined how dependent actor assignments can be supported adequately by a WfMS. With dependent actor assignments, the possible actors of an activity depend on previous activities. This is a practically very relevant case, which has not been considered in the WF literature so far. We have shown how suitable load estimations can be already performed at build time. Additionally, we have shown how the WF control can be realized in a way that the WF server of an activity can be chosen dynamically at run time. For this purpose, we have introduced variable server assignment expressions as well as models for probability and cost estimations. The cost model and the distribution algorithms were implemented and measurements were performed in order to show their correctness [14]. In addition, the effectiveness of the techniques presented was confirmed by simulations. Distributed WF execution (inclusive variable server assignments) was realized in the ADEPT-WfMS. Since the whole system is implemented in Java, it can be used in the Internet.

To get a complete picture, several additional aspects have to be considered as well (c.f. [18]). Transactional aspects, for example, may influence the communication load. Due to lack of space we could not discuss that in this paper. Our analysis performed so far make us confident that our approach suffers not much more (if at all) than most of the other distributed approaches.

Acknowledgements: We would like to thank our colleagues Manfred Reichert and Clemens Hensinger for their valuable suggestions.

References

1. G. Alonso, M. Kamath, D. Agrawal, A. El Abbadi, R. Günthör, and C. Mohan: *Failure Handling in Large Scale Workflow Management Systems.* Technical Report RJ9913, IBM Almaden Research Center, 1994.
2. G. Alonso, C. Mohan, R. Günthör, D. Agrawal, A. El Abbadi, and M. Kamath: *Exotica/FMQM: A Persistent Message-Based Architecture for Distributed Workflow Management.* Proc. IFIP Working Conf. on Information Systems for Decentralized Organisations, Trondheim, 1995.
3. T. Bauer and P. Dadam: *A Distributed Execution Environment for Large-Scale Workflow Mana-*

gement Systems with Subnets and Server Migration. Proc. 2nd IFCIS Conf. on Cooperative Information Systems, pages 99-108, Kiawah Island, SC, 1997.
4. T. Bauer and P. Dadam: *Architectures for Scalable Workflow Management Systems – Classification and Analysis*. Technical Report UIB 98-02, Universität Ulm, Fakultät für Informatik, 1998 (in German).
5. T. Bauer and P. Dadam: *Distribution Models for Workflow Management Systems – Classification and Simulation.* Informatik Forschung und Entwicklung, 14(4):203-217, 1999 (in German).
6. T. Bauer and P. Dadam: *Efficient Distributed Control of Enterprise-Wide and Cross-Enterprise Workflows.* In: Proc. Workshop Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications, 29. Jahrestagung der GI, pages 25-32, Paderborn, 1999.
7. T. Bauer and P. Dadam: *Variable Server Assignments and Complex Actor Assignments in the ADEPT Workflow Management System.* Technical Report UIB 2000-02, Universität Ulm, Fakultät für Informatik, 2000 (in German).
8. D. Barbará, S. Mehrotra, and M. Rusinkiewicz: *INCAs: Managing Dynamic Workflows in Distributed Environments.* Journal of Database Management, Special Issue on Multidatabases, 7(1):5-15, 1996.
9. S. Ceri, P. Grefen, and G. Sánchez: *WIDE – A Distributed Architecture for Workflow Management.* 7th Int. Workshop on Research Issues in Data Engineering, Birmingham, 1997.
10. U. Dayal, M. Hsu, and R. Ladin: *A Transactional Model for Long-Running Activities.* Proc. 17th VLDB, pages 113-122, Barcelona, 1991.
11. S. Das, K. Kochut, J. Miller, A. Sheth, and D. Worah: *ORBWork: A Reliable Distributed CORBA-based Workflow Enactment System for METEOR$_2$.* Technical Report #UGA-CS-TR-97-001, Department of Computer Science, University of Georgia, 1997.
12. A. Dogac et al: *Design and Implementation of a Distributed Workflow Management System: METUFlow.* Proc. NATO Advanced Study Institute on Workflow Management Systems and Interoperability, pages 61-91, Istanbul, 1997.
13. J. Eder, H. Groiss, and W. Liebhart: *Workflow Management and Databases.* Proc. 2ème Forum Int. d´Informatique Appliquée, Tunis, 1996.
14. H. Enderlin: *Realization of a Distributed Workflow Execution Component on Basis of IBM FlowMark.* Master's thesis, Universität Ulm, Fakultät für Informatik, 1998 (in German).
15. P. Heinl and H. Schuster: *Towards a Highly Scaleable Architecture for Workflow Management Systems.* Proc. 7th Int. Workshop on Database and Expert Systems Applications, pages 439-444, Zürich, 1996.
16. P. Muth, D. Wodtke, J. Weißenfels, A. Kotz-Dittrich, and G. Weikum: *From Centralized Workflow Specification to Distributed Workflow Execution.* Journal of Intelligent Information Systems, 10(2):159-184, 1998.
17. G. Piccinelli: *Distributed Workflow Management: The TEAM Model.* Proc. 3nd IFCIS Conf. on Cooperative Information Systems, pages 292-299, New York, 1998.
18. M. Reichert, T. Bauer and P. Dadam: *Enterprise-Wide and Cross-Enterprise Workflow-Management: Challenges and Research Issues for Adaptive Workflows.* In: Proc. Workshop Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications, 29. Jahrestagung der GI, pages 56-64, Paderborn, 1999.
19. M. Reichert and P. Dadam: *ADEPT$_{flex}$ – Supporting Dynamic Changes of Workflows Without Losing Control*. Journal of Intelligent Information Systems, 10(2):93-129, 1998.
20. A. Schill and C. Mittasch: *Workflow Management Systems on Top of OSF DCE and OMG CORBA.* Distributed Systems Engineering, 3(4):250-262, 1996.
21. H. Schuster, J. Neeb, and R. Schamburger: *A Configuration Management Approach for Large Workflow Management Systems.* Proc. Joint Conf. on Work Activities Coordination and Collaboration, San Francisco, 1999.
22. G. Vossen, M. Weske, and G. Wittowski: *Dynamic Workflow Management on the Web.* Technical Report 24/96-I, Lehrstuhl für Informatik, Universität Münster, 1996.