DYNAMIC CHANGE OF SERVER ASSIGNMENTS IN DISTRIBUTED WORKFLOW MANAGEMENT SYSTEMS

Thomas Bauer
DaimlerChrysler Research and Technology, RIC/ED
P.O. Box 2360, D-89013 Ulm, Germany

Manfred Reichert
University of Ulm, Dept. Databases and Information Systems
James-Franck-Ring, D-89069 Ulm, Germany

Keywords: Workflow Management System, Scalability, Distributed Workflow Management, Dynamic Server Assignment

Abstract: Workflow management systems (WfMS) offer a promising approach for realizing process-oriented information systems. Central WfMS, with a single server controlling all workflow (WF) instances, however, may become overloaded very soon. In the literature, therefore, many approaches suggest using a multi-server WfMS with distributed WF control. In such a distributed WfMS, the concrete WF server for the control of a particular WF activity is usually defined by an associated server assignment. Following such a partitioning approach, problems may occur if components (WF servers, subnets, or gateways) become overloaded or break down. As we know from other fields of computer science, a favorable approach to handle such cases may be to dynamically change hardware assignment. This corresponds to the dynamic change of server assignments in WfMS. This paper analyses to what extend this approach is reasonable in such situations.

1 INTRODUCTION

WfMS allow computerized business processes to be run in a distributed system environment (Leymann and Roller, 2000). The bottom line for any effective WfMS is to enable easy development and maintainance of large process-oriented application systems. For this purpose, application-specific code of single process steps is separated from the flow logic of respective business processes. So instead of a large, monolithic program package, we obtain a set of individual activities which represent the application programs. Their orchestration (i.e., the process logic), however, is specified in a separate control flow definition that sets out the order and conditions according to which the individual activities are to be executed. At runtime, the WfMS takes care that activities of a WF instance are executed according to pre-defined control flow. When a certain activity becomes executable, it is inserted into worklists of authorized actors. Exactly which user is authorized to work on this activity is generally determined by the user role assigned to it. Generally, it is possible that several actors are allowed to perform a certain activity.

(Central) WfMS, in general, have the disadvantage that their single WF server may become overloaded due to the large number of WF instances it has to control and due to its numerous tasks (e.g., refreshing worklists, calculating the new state of a WF instance after activity completion, starting activity programs, etc.). To avoid overload situations, different approaches for distributed WF management were developed in the past (Bauer, 2001; Kochut et al., 2003; Muth et al., 1998; Schuster et al., 1999). All of them follow the idea to distribute the overall system load to several servers. In doing so, normally, for each WF activity they try to choose a well-suited WF server for the control of this activity. For this purpose, for example, the server of the subnet containing the greatest number of potential actors of the respective activity may be used. This strategy results in a small distance between the WF server and the clients with respect to network topology and, therefore, in a good communication behavior (short response times and low communication costs) (Bauer, 2001; Bauer et al., 2003).

For approaches dealing with distributed WF management, usually, the WF server controlling a particular WF activity is already determined at buildtime (Bauer, 2001; Muth et al., 1998; Das et al., 1997). In certain situations (e.g., server overload or breakdown), however, it may be advantageous to deviate from the pre-planned server assignment during runtime. A method to deal with such cases, which is well known from other fields of computer science, is

Bauer T. and Reichert M. (2004).
DYNAMIC CHANGE OF SERVER ASSIGNMENTS IN DISTRIBUTED WORKFLOW MANAGEMENT SYSTEMS.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 91-98
DOI: 10.5220/0002628900910098

to assign tasks dynamically to the hardware components. It has been applied, for example, for scheduling processes in operating systems (Casavant and Kuhl, 1988; Goscinski, 1991), where the processor executing a particular operating system process is usually selected when this process is started. As a second example consider CORBA (OMG, 1995) where the execution server for a method call is selected at runtime as well. In both approaches, thereby, the current load and failure situation may be taken into account.

This commonly used approach may be applied to distributed WfMS as well. One goal is to compensate overloading of WfMS components (WF server, subnet, or gateway). For example, if a specific WF server becomes overloaded, tasks it was originally intended for may be assigned to another server with lower (current) load. As another scenario, in which the dynamic change of server assignments – in the following called *dynamic server changes* for short – is reasonable, consider the breakdown of a WfMS component. In such a case, a different WF server may be chosen (dynamically) for the control of the affected activities.

At first glance, support of dynamic server changes seems to be advantageous. We, therefore, examine whether they can be used to solve the overload and failure problems in distributed WfMS. However, our analysis will show that dynamic server changes are not appropriate to solve these problems. The main contribution of this paper is to explain the difficulties, occurring in this context, in detail. In addition, we describe ways to counteract overload and breakdown of WfMS components in a more suitable manner.

The next section shortly introduces important fundamentals of distributed WF management. They are necessary for the further understanding of this paper. Section 3 examines in which states of a WF instance the dynamic change of server assignments is possible and does also make sense. Section 4 addresses the treatment of overload situations by dynamically changing server assignments. Section 5 discusses the breakdown of components. In Section 6, alternatives to dynamic server changes are presented, which are more suitable to solve the problems described above. Related approaches are discussed in Section 7. The paper closes with a summary of the results.

2 DISTRIBUTED WORKFLOW MANAGEMENT

We consider both enterprise-wide and cross-organizational, process-oriented applications in operative use. For example, (Kamath et al., 1996; Sheth and Kochut, 1997) deal with applications leading to a WfMS with more than 10000 users and a number of currently active WF instances achieving the same magnitude. Owing to this large number of users and co-active WF instances, the WfMS is generally subjected to an extremely heavy load. Single components of the system, therefore, may become quickly overloaded.

To avoid overload situations, several approaches suggest to not only control a WF instance by a single WF server. Instead, its WF schema is partitioned and the resulting partitions are executed by different WF servers (Bauer and Dadam, 1997) (see Fig. 1).

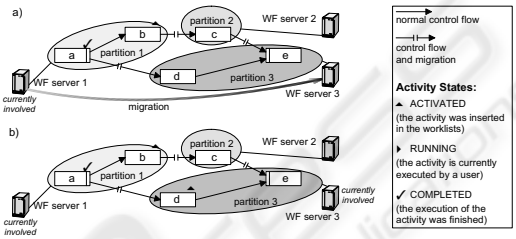

Figure 1: a) Migration of a WF instance (from server 1 to server 3) and b) the resulting state of the WF instance.

This also applies to ADEPT$_{distribution}$ (Bauer, 2001; Bauer et al., 2003), the distributed variant of the ADEPT[1] approach (Dadam et al., 2000; Reichert and Dadam, 1998). When the end of a partition is reached during runtime, control over the respective WF instance is handed over to the next WF server. To perform such a *migration*, a description of the WF instance state (current execution state and values of parameter data of activity programs) has to be transferred to the target server before it can take over control (Bauer et al., 2001).

As already mentioned, partitioning of WF schemes at buildtime and distributed WF control during runtime have been successfully utilized by other approaches as well (Casati et al., 1996; Muth et al., 1998). Some of them also aim at minimizing the overall communication costs. The experiences we gained with legacy WfMS have shown that there is a great deal of communication between a WF server and its clients, oftentimes necessitating the exchange of large amounts of data. This, in turn, may overload the communication system as well. In ADEPT$_{distribution}$, for WF activities the controlling servers are automatically calculated at buildtime. This is done in a way that minimizes the overall communication (Bauer and Dadam, 1997; Bauer and Dadam, 2000). In doing so, typically, the WF server selected to control a specific activity will reside in that subnet the greatest number of potential actors belongs to.

[1] Application Development Based on Encapsulated Pre-Modeled Process Templates.

3 APPROACHES FOR DYNAMIC SERVER CHANGES

The way of how to perform dynamic server changes significantly influences resulting costs and benefits. In this section, we examine general approaches for migrating a WF instance to a target server, which is dynamically selected at runtime (*dynamic migration*).

Generally, reaction on overload or failure situations should happen very quickly. For concerned WF instances, this requires to dynamically change the WF server of currently executed activities. In order to gain larger impact and to prevent unnecessary migrations, it is always recommendable to change the assignment of the whole current partition. A vital aspect, thereby, is the point in time this happens. Generally, the following possibilities exist (cf. Fig. 2):

Approach 1: Dynamic Server Changes May Happen at Any Point in Time
Dynamic changes of server assignments are allowed at any time during WF execution. As a consequence the originally intended server has to notify all clients involved in the processing of the current activity about the change. Otherwise, responses would be directed to the wrong (namely to the old) WF server. Apart from this, normally, this approach requires an additional migration when a server assignment is changed.

Approach 2: Dynamic Server Changes are Prohibited for Activities to be Started
Dynamic server changes are forbidden if the activity (of the concerned execution branch) has currently state ACTIVATED (Reichert and Dadam, 1998). During this state, corresponding work items are offered to all potential actors who may work on this activity. In case of dynamic migrations at most one client has to be informed about the server change, namely the client of that user actually executing this activity.

Approach 3: Dynamic Server Changes are Only Allowed After Completion of Activities
Dynamic migrations are only performed if a transition between two consecutive activities currently takes place. At this time, no client is involved in processing the corresponding execution branch of the WF instance. (The completed activity is not contained in worklists any more, it is currently not executed, and there are no work items belonging to the subsequent activity yet.) Clients, therefore, need not be informed about the server change; there is even no need for them to know that dynamic server changes are possible. Using this approach, solely an additional migration may have to be performed in order to realize the dynamic server change.

Approach 4: Server Changes are Allowed Only in Case of Pre-planned Migrations
Dynamic server changes are only possible if a pre-planned migration currently takes place.[2] In doing so, only the target server of a (pending) migration has to be changed. This approach does not require any additional migration.

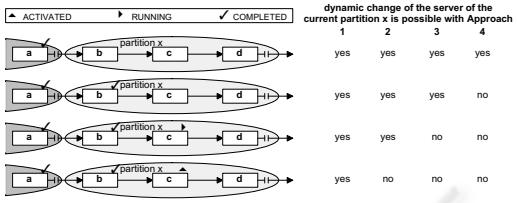

Figure 2: Possibilities for realizing dynamic server changes.

Let us shortly discuss the efficiency of these four approaches: Realization of Approaches 1 and 2 is very expensive since clients have to be involved in dynamic server changes. In addition, they require much communication effort and are error-prone. Errors may occur, for example, if a client, which could temporarily not reach its server, tries to transmit the output data of an activity program to this (already obsolete) WF server. Approach 3 creates communication load as well since additional migrations are required for dynamically changing a server. Only Approach 4 does not require additional communication and does not have any impact on clients. Using this approach, however, less WF instances may be subject of a dynamic server change (because of their current state). Reason is that only such WF instances may be considered, for which a (pre-planned) migration currently takes place. Nevertheless, there is still a large number of WF instances falling into this category. Note that in enterprise-wide scenarios a WF server controls many WF instances at each point in time. This is especially valid for high-performance WfMS as considered in this paper. It is no good idea, anyway, to migrate too many WF instances at the same time since this may lead to the underload of the formerly overloaded server; i.e., to a "flattering" of the load.

To avoid unnecessary limitations, in the following, all approaches are considered despite of the weaknesses discussed. It is examined, thereby, which approach is most suited for solving the given problems.

4 TREATMENT OF OVERLOAD SITUATIONS

In the following, we examine to what extend Approaches 1-4 are suited to compensate the overload of WfMS components. At first, we consider the overload

[2] The creation of a new WF instance is treated similar to a migration. Therefore, the first partition of a WF instance may be subject of a dynamic server change as well.

of communication system components (subnets, gateways). Then we inspect overloading of WF servers. Finally, we discuss general issues related to dynamic server changes in the context of overload situations.

4.1 Overloaded Components of the Communication System

4.1.1 Overloaded Subnets

Let us assume the scenario depicted in Fig. 3, where subnet x of the WfMS is currently overloaded. This overload shall be compensated by assigning WF instances, normally controlled by the WF server of subnet x, to another server.

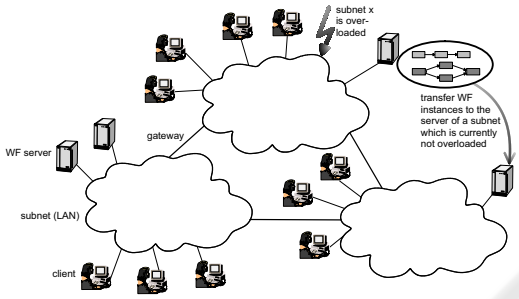

Figure 3: Dynamic change of server assignments in the case of subnet overload.

Unfortunately, dynamic server changes will not contribute to reduce the load of subnet x if the majority of actors who may work on respective activities belongs to subnet x as well. The communication between the WF clients of these users and the WF server will continue crossing subnet x, since the clients will still be located in this subnet. The overloaded subnet, therefore, will not be significantly unburdened.

However, such an actor distribution is the normal case, since the majority of potential actors is located in the subnet of the WF server controlling the current partition. Note that this server has just been chosen because of this actor distribution. This makes sense since such a server assignment contributes to reduce the overall communication costs (cf. Section 2). The usage of dynamic migrations, however, has a negative effect on this goal.

If one of the Approaches 1-3, as described in Section 3, is used to realize dynamic server changes, the total situation becomes even worse due to the additionally required migration. Such a migration always burdens the subnet to which the source server belongs. The usage of Approaches 1 or 2 makes the load situation even more worse. For these cases, a dynamic server change further necessitates additional communications with clients.

4.1.2 Overloaded Gateways

The overload of a gateway (e.g., a long distance communication link) may be handled in a similar way: WF instances may be dynamically migrated to a WF server that belongs to a subnet not connected with this gateway. Using this method, however, similar problems as described in the previous section occur. Therefore, this approach is not recommendable. It is even less advantageous when compared to treatment of overloaded subnets since the subnet of the originally intended WF server may be connected to several gateways. All of them are unburdened by the dynamic migration. The load reduction of the overloaded gateway, therefore, only happens proportionately.

4.2 Overloaded Workflow Servers

The following possibilities exist to reduce the load of an overloaded WF server: WF instances may be "migrated away" from it or instances, it was originally intended for, are not transferred to this server.

In this scenario, dynamic server changes more likely lead to the intended goal because the WF server has to control fewer WF instances in fact. Using Approaches 1-3 of Section 3, however, an overloaded WF server may be further burdened due to the migrations now required. Since the effort necessary to handle such migrations is larger than for regular activity executions (because of the larger data volume, cf. (Bauer et al., 2001)), the suggested method may even be disadvantageous. With Approaches 1 and 2, in addition, the WF server has to handle interactions with clients when performing dynamic migrations. This results in additional effort. Furthermore, the general disadvantages of dynamic server changes with respect to compensation of overload situations have negative effects on the quality of the presented approach (see the discussion in the following section).

4.3 Dynamic Server Changes: General Issues

We have sketched methods for dynamic server changes in order to handle overload situations of both the communication system and the WF servers. In the following, we discuss general difficulties arising in this context.

Just as all methods for load balancing, the presented approaches require exchange of load information (in any kind). First of all, thereby, it is not relevant which technique is actually used for this purpose. Numerous variants are known from distributed scheduling of operating systems processes (Goscinski, 1991). It is possible, for example, to exchange load information periodically between the WF

servers. Another possibility is to request the current load from potential target servers when a dynamic migration has to be performed. All these methods have in common that the usage of dynamic server changes itself burdens the components of the WfMS and, therefore, increases the risk of an overload.

This disadvantage, however, can be weakened by using "piggybacking" (Tanenbaum, 1996): The (relative small) load information to be exchanged is transmitted together with other communications, which are necessary anyway. If it can be supposed that such communications (e.g., migrations) occur between all pairs of WF servers in a sufficient frequency, no additional messages become necessary for the exchange of load information. Only the data volume to be transmitted will be (marginally) increased.

A more crucial disadvantage of load-dependent, dynamic server changes results from the long "pauses", which often occur during WF execution. On the one hand, they may be caused by the priorities the users possibly set; on the other hand, long-running processes may comprise active steps (e.g. medications in a chemotherapy process) resting for days or weeks, before they are allowed to be executed. For this reason, a WF instance, which was dynamically migrated to unburden a component, may rest for a long time. In such a case, in the short term, we do not achieve any load reduction at all. Instead the load reduction for the component happens much later. But then, it may even be underloaded.

If one of the Approaches 1-3 is used, the situation is even worse: The migration necessary for the dynamic server change is performed immediately. This results in additional effort at a point in time the overload still exists. With Approaches 1 and 2, the same applies for the interaction with the clients as well.

4.4 Assessment

Dynamic server changes do not seem to be suitable to deal with overloaded communication system components. Regarding WF servers, a load reduction is theoretically possible, but occurs with a considerable delay. If such a method is used, however, it is crucial that Approach 4 (cf. Section 3) is applied in order to avoid additional load for the WF server.

5 TREATMENT OF FAILURE SITUATIONS

To deal with the breakdown of system components the following method is conceivable:(Partitions of) WF instances affected by the failure are dynamically assigned to another WF server. As we will see, in this context it does not matter whether the breakdown of a WF server or a communication system component has to be compensated. In the following, therefore, we do not distinguish between these two cases.

5.1 Using Dynamic Server Changes to Compensate Breakdowns

Assume that the breakdown of a WfMS component shall be compensated by dynamically changing server assignments. In this case, the concerned WF instances cannot be "migrated away" from their current server; i.e., Approaches 1-3 have to be ruled out. Such a migration would just require the operativeness of the failed component. This is obvious for WF servers as well as for subnets since these components would always be directly involved in the corresponding migration.

The breakdown of a gateway is only relevant if this leads to the fragmentation of the communication network. Otherwise, it would be still possible for the WF server to reach all of its clients. In case of a network fragmentation, however, a dynamic migration to the other fragment of the network would be required. This is the only possibility for the WF server to reach the clients located there. Such a migration, however, is impossible due to the breakdown of the communication link.

For compensating component failures, therefore, only Approach 4 can be used. With such an approach, however, it is not possible to improve the situation for users who have to wait for the processing of their actions (due to the component failure). Neither they can start activities offered to them in their worklists nor they can complete activities they are currently working on. Reason is that the corresponding WF server is currently not accessible. These serious problems are visible to the end user and cannot be solved by dynamic server changes.

In failure situations, usage of Approach 4 has the consequence that another target server may be chosen for a pending migration of a WF instance. This, however, is not very helpful since in this state no users are waiting for an action belonging to this WF instance. (Note that the WF instance currently migrates from one server to another; i.e., the previous activity was completed and the succeeding activity has not yet been activated.) The serious problems, described in the previous paragraph, do not occur in this situation anyway. In addition, it can be assumed that a failed component will be repaired early. The delay (that is not visible to the users), therefore, does not result in a problem that, in turn, would justify the additional effort as described.

5.2 Assessment

As described above, dynamic server changes are not well suited to compensate the breakdown of WfMS components. There are methods, however, which are much more suitable for this purpose; e.g. the use of backup servers (Kamath et al., 1996). Corresponding approaches as well as methods for the treatment of overload problems are discussed in Section 6.

Since the shortcomings of dynamic server changes are obvious, we omit a formal presentation of the sketched methods at this point. The same applies to the comparison of possible realization variants (e.g., techniques for the selection of the target server of a dynamic migration; i.e., the question, which server assignment is concretely chosen). It does also not make much sense to deal with methods for the (efficient) recognition of component failures. (The same applies to methods for the efficient realization of the load information exchange and criteria at which load and how long one wants to react on overload situations.) Finally, it is superfluous to evaluate the quality of the corresponding methods by a model calculation or simulation, because their flaws are obvious.

6 ALTERNATIVE SOLUTIONS

Dynamic server changes have turned out to be unsuitable to solve the overload and failure problems coming up with enterprise-wide applications. In the following, we sketch some more promising approaches. Most of them have been addressed and implemented within the ADEPT project (Reichert et al., 2003).

In the ADEPT WfMS, the prevention of communication system overloads has been a crucial aspect from the very beginning. We developed advanced methods, which allow reducing the data volume communicated in a WfMS. These methods may be used in combination with each other as well.

We developed a sophisticated method to calculate optimal server assignments already at buildtime (Bauer and Dadam, 1997). They result in minimized total communication costs at runtime. For this purpose, we developed algorithms which estimate the communication costs occurring during WF execution. This information is used for calculating optimal server assignments. Our algorithms are based on a realistic cost model and make use of both process and organization model data (Bauer, 2001).

Variable server assignments can be used to reduce transmitted data volumes (Bauer and Dadam, 1999) if concrete actors of an activity depend on preceding steps (e.g., if an activity shall be performed by the same user as a preceding one). In such cases, more flexible server assignments can be used such that the server to be actually chosen may depend on preceding activities; i.e., the server for controlling activities of a certain WF partition needs not always be determined at buildtime. A detailed description of this approach can be found in (Bauer and Dadam, 2000).

In (Bauer et al., 2001) we present methods which can be used to significantly reduce the data volumes transmitted during migrations. In particular, they prevent redundant transfer of WF instance data to the same server. Such redundant transmissions may occur in connection with repeated migrations to the same server (e.g., due to loop backs).

In some cases, even these measures may be insufficient to prevent communication system overloading. Then smaller subnets must be created, of which each has to serve a smaller number of users.

The problem of overloaded servers has been considered in the ADEPT project as well. In (Bauer et al., 2003), for example, we present an advanced method which can be used to prevent overloading of WF servers. It enables the usage of an arbitrary number of WF servers in the same subnet. In addition, the load can be distributed to these servers in an arbitrary and definable ratio. Since our method does not require any additional communication for WF instance execution, there is no risk for the communication system to become overloaded.

If very high availability is required, usage of redundant hardware will be the best way to compensate server or communication system failures. (Kamath et al., 1996) presents methods for backup servers. The information necessary for WF instance control is continuously transmitted from the processing to the backup server. In case of server breakdown, therefore, the backup server is able to take over control.

Finally, failures of single components of the communication system can be compensated by using redundant communication hardware. This may be implemented, for example, by double communication rings FDDI (Tanenbaum, 1996).

7 RELATED WORK

Methods which make use of load information have been applied to the scheduling of operating system processes for a long time and with great success (Casavant and Kuhl, 1988; Goscinski, 1991). Based on current load situation, they determine the processor to which a certain process shall be assigned. This processor, however, is normally determined when starting an operating system process (non-preemptive scheduling (Casavant and Kuhl, 1988)). Consequently, no running processes have to be transferred.(This would correspond to a migration.)

In this context, it is important to mention that op-

erating system processes do not correspond to WF instances of WfMS. A WF instance consists of activity instances, which have to be assigned to one or multiple WF servers. In the same way, an application consists of operation system processes, which are assigned to the processors. Consequently, the operating system processes correspond to the activity instances. When assigning activity instances to WF servers, useful information about the activities is available (e.g., about actor assignments and distribution of users to subnets). This, however, is normally not the case for operating system processes. When scheduling WF instances, therefore, it is possible to select a much more favorable WF server, with respect to the communication behavior, as this would be possible by pure load balancing. The usage of such information corresponds to the application of static scheduling methods (Casavant and Kuhl, 1988) in operating systems, instead of the usually applied dynamic methods.

In the following, approaches for distributed WF management are discussed in the context of dynamic server changes (see also (Bauer, 2001; Bauer and Dadam, 1999)). Comparable to ADEPT, most of them determine server assignments for activity instances based on process and organization model data. Examples are MENTOR (Muth et al., 1998) and WIDE (Casati et al., 1996). Both allocate the WF server for an activity close to its potential actors. The same applies to MOBILE (Heinl and Schuster, 1996). As opposed to the previous approaches, however, in MOBILE a WF instance is always controlled by the same server during its whole lifetime. Consequently, no migrations become necessary. Following this approach, however, a sub-process may be controlled by another WF server (Schuster et al., 1999). It is selected at runtime considering various criteria like access rights or weights. METEOR$_2$ (Das et al., 1997; Kochut et al., 2003), CodAlf and BPAFrame (both (Schill and Mittasch, 1996)) always chose the WF server near to the application that belongs to the current activity. Finally, there are completely distributed approaches (e.g., Exotica/FMQM (Alonso et al., 1995) and INCAS (Barbará et al., 1996)). For them, usually the machine of the current actor carries out the server function. – All these approaches have in common that the scheduling of activity instances is performed independently of the current load and failure situation.

The Exotica/Cluster approach (Alonso et al., 1994) shows that it is possible to realize scheduling in WfMS independently of its process and organization models. Using this approach, the WF server (cluster) of a WF instance is selected randomly at creation time. The WF instance remains in this cluster during its whole lifetime; i.e., no migrations are required. This approach, however, does also not use any load information for the selection of the WF server.(But it can be easily modified this way.) In addition, (Bauer, 2001) shows that the non-usage of model data results in an unfavorable communication behavior.

CORBA-based approaches for distributed WF management have been presented in the literature as well: WASA$_2$ (Weske, 1999) considers activities to be CORBA objects which signalize state changes to each other. Since these objects may be allocated to arbitrary computers, distributed WF management can be realized this way. The distribution model of MOKASSIN (Joeris and Herzog, 1999) is similar to the previous approach: The activity instances are CORBA objects. They use events to communicate with each other and they can be placed on arbitrary computers. With these approaches, in principle, the CORBA middleware enables the realization of arbitrary distribution. It would be possible, thereby, to realize the distribution in a way that considers load and failure information. With such a method, however, the difficulties already discussed would occur.

8 SUMMARY

We have analyzed to what extend dynamic changes of server assignments are useful in order to react to overload situations and component failures in a distributed WfMS. Our analyses have shown that such dynamic migrations are not suitable to deal with overload situations. In most cases, they enable almost no load reduction for subnets and gateways. The required additional effort may even have the consequence that the total situation becomes worse. Only for WF servers, it is possible to reduce the load, but this reduction is very limited. For the compensation of WfMS component failures dynamic server changes are unsuited as well. The impacts of such breakdowns, which are really critical for the end user, cannot be reduced by a dynamic migration. Reason is that the breakdown of the WfMS component prevents the execution of the dynamic migrations. To sum it up, the usage of dynamic server changes is not recommendable.

We have sketched alternative methods, which we realized in the ADEPT project. However, it was not the goal to discuss them in detail (this happened in other publications). Instead, we only want to show that there are alternatives to the usage of dynamic server changes. With these methods, it becomes possible to compensate the overload of both the communication system and the WF servers. In addition, they can handle failures of these WfMS components.

REFERENCES

Alonso, G., Kamath, M., Agrawal, D., El Abbadi, A., Günthör, R., and Mohan, C. (1994). Failure Han-

dling in Large Scale WfMS. Technical report, IBM Almaden Research Center.

Alonso, G., Mohan, C., Günthör, R., Agrawal, D., El Abbadi, A., and Kamath, M. (1995). Exotica/FMQM: A Persistent Message-Based Architecture for Distributed Workflow Management. In *Proc. IFIP Working Conf. on Inf. Syst. for Decentralized Organisations*, Trondheim.

Barbará, D., Mehrotra, S., and Rusinkiewicz, M. (1996). INCAs: Managing Dynamic Workflows in Distributed Environments. *Journal of Database Management*, 7(1).

Bauer, T. (2001). *Efficient Realization of Enterprise-wide WfMS*. PhD thesis, University of Ulm. (Tenea-Verlag, in German).

Bauer, T. and Dadam, P. (1997). A Distributed Execution Environment for Large-Scale WfMS with Subnets and Server Migration. In *Proc. 2nd IFCIS Conf. on Coop. Inf. Syst.*, Kiawah Island.

Bauer, T. and Dadam, P. (1999). Distribution Models for WfMS - Classification and Simulation. *Informatik Forschung und Entwicklung*, 14(4).

Bauer, T. and Dadam, P. (2000). Efficient Distributed Workflow Management Based on Variable Server Assignments. In *Proc. 12th Conf. on Advanced Inf. Syst. Engineering*, Stockholm.

Bauer, T., Reichert, M., and Dadam, P. (2001). Efficient Transmission of Process Instance Data in Distributed WfMS. *Informatik Forschung und Entwicklung*, 16(2). (in German).

Bauer, T., Reichert, M., and Dadam, P. (2003). Intra-Subnet Load Balancing in Distributed WfMS. *Int. J Cooperative Information Systems*, 12(3).

Casati, F., Grefen, P., Pernici, B., Pozzi, G., and Sánchez, G. (1996). WIDE: Workflow Model and Architecture. CTIT Technical Report 96-19, University of Twente.

Casavant, T. and Kuhl, J. (1988). A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems. *IEEE Transactions on Software Engineering*, 14(2):141–154.

Dadam, P., Reichert, M., and Kuhn, K. (2000). Clinical Workflows - The Killer Application for Process-oriented Information Systems? In *Proc. 4th Int. Conf. on Business Inf. Syst.*, Posen.

Das, S., Kochut, K., Miller, J., Sheth, A., and Worah, D. (1997). ORBWork: A Reliable Distributed CORBA-based Workflow Enactment System for METEOR$_2$. Technical Report, Department of Computer Science, University of Georgia.

Goscinski, A. (1991). *Distributed Operating Systems: The Logical Design*. Addison-Wesley.

Heinl, P. and Schuster, H. (1996). Towards a Highly Scaleable Architecture for WfMS. In *Proc. 7th Int. Workshop on Database and Expert Systems Applications*, Zurich.

Joeris, G. and Herzog, O. (1999). Towards Flexible and High-Level Modeling and Enacting of Processes. In *Proc. 11th Int. Conf. on Advanced Information Systems Engineering*, Heidelberg.

Kamath, M., Alonso, G., Günthör, R., and Mohan, C. (1996). Providing High Availability in Very Large WfMS. In *Proc. 5th Int. Conf. on Extending Database Technology*, Avignon.

Kochut, K., Arnold, J., Sheth, A., Miller, J., Kraemer, E., Arpinar, B., and Cardoso, J. (2003). IntelligGEN: A Distributed Workflow System for Discovering Protein-Protein Interactions. *Distributed and Parallel Databases*, 13:43–72.

Leymann, F. and Roller, D. (2000). *Production Workflow - Concepts and Techniques*. Prentice Hall.

Muth, P., Wodtke, D., Weißenfels, J., Kotz-Dittrich, A., and Weikum, G. (1998). From Centralized Workflow Specification to Distributed Workflow Execution. *JIIS*, 10(2).

OMG (1995). The Common Object Request Broker: Architecture and Specification. Technical Report Revision 2.0, Object Management Group.

Reichert, M. and Dadam, P. (1998). ADEPT$_{flex}$ – Supporting Dynamic Changes of Workflows Without Losing Control. *JIIS*, 10(2).

Reichert, M., Rinderle, S., and Dadam, P. (2003). ADEPT Workflow Management System: Flexible Support For Enterprise-wide Business Processes. In *Proc. Int'l Conf. Business Process Management (BPM'03)*, Eindhoven.

Schill, A. and Mittasch, C. (1996). WfMS on Top of OSF DCE and OMG CORBA. *Distributed Systems Engineering*, 3(4).

Schuster, H., Neeb, J., and Schamburger, R. (1999). A Configuration Management Approach for Large WfMS. In *Proc. Int. Conf. on Work Activities Coordination and Collaboration*, San Francisco.

Sheth, A. and Kochut, K. (1997). Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems. In *Proc. NATO Advanced Study Institute on WfMS and Interoperability*, Istanbul.

Tanenbaum, A. (1996). *Computer Networks*. Prentice-Hall.

Weske, M. (1999). Workflow Management Through Distributed and Persistent CORBA Workflow Objects. In *Proc. 11th Int. Conf. on Advanced Information Systems Engineering*, Heidelberg.