Towards Runtime Migration of WS-BPEL Processes*

Sonja Zaplata, Kristian Kottke, Matthias Meiners, and Winfried Lamersdorf

Distributed Systems and Information Systems
Computer Science Department, University of Hamburg
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
{zaplata,3kottke,4meiners,lamersd}@informatik.uni-hamburg.de

Abstract. The decentralized execution of business process instances is a promising approach for enabling flexible reactions to contextual changes at runtime. Most current approaches address such process distribution by *physical fragmentation* of processes and by dynamic assignment of resulting static process parts to different business partners.

In order to enable a more dynamic segmentation of such responsibilities at runtime, this paper proposes to use *process runtime migration* as a means of *logical process fragmentation*. Accordingly, the paper presents a general migration metadata model and a corresponding basic privacy and security mechanism for enhancing existing process models with the ability for runtime migration while respecting the intensions and privacy requirements of both process modelers and initiators. The approach is conceptually evaluated by applying it to WS-BPEL processes and comparing the results to the general concept of process fragmentation.

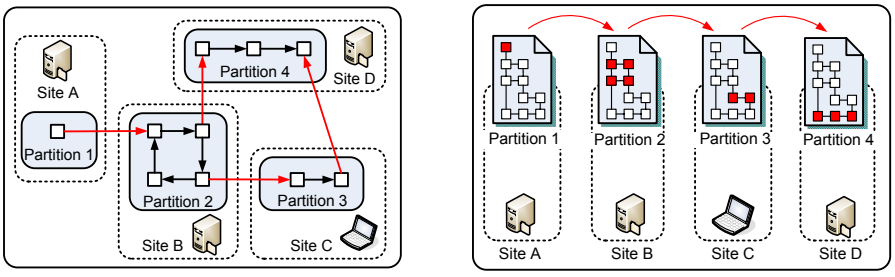
1 Motivation

In todays networked business environments, cross-organizational collaborations composing complementary services and thus realizing new, value-added products gain increasing importance. As a technical representation of such business processes, executable workflows allow for flexible, dynamic and loosely-coupled collaboration among several business partners. The *Business Process Execution Language for Web Services (WS-BPEL)*[1] is currently one of the most relevant practical approaches. It allows for distributing resources such as employees, machines and services, whereas process control flow logic is typically executed by one single component at one single site [2].

However, due to the autonomy of participants, a single centralized process management system to control the execution of cross-organizational processes is often neither technically nor organizationally desired. As an example, required services and resources often cannot be accessed by a centralized process engine

* The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

A. Dan, F. Gittler, and F. Toumani (Eds.): ICSOC/ServiceWave 2009, LNCS 6275, pp. 477–487, 2010.



(a) Process fragmentation (b) Process migration

Fig. 1. Process decentralization variants

because of technological differences or due to security policies [3]. Furthermore, in some cases the location where a process fragment is executed is relevant to perform the required functionality or is necessary for judicial reasons, e.g. in the context of eGovernment. Related to this, other non-functional aspects such as execution time, performance, navigation cost and capacity utilization can be optimized by load balancing and thus improve flexibility and scalability of participating systems [4].

Most current research is approaching decentralization of control flow navigation by a *physical fragmentation* of processes – splitting the overall executable process into several subparts which are then distributed to a number of available process engines (cp. figure 1(a)). In contrast to that, this paper proposes *process instance migration* as a means of *logical fragmentation*, fragmenting only the responsibilities for the execution of the process into a set of sub-responsibilities while preserving the original structure of the process description for all of the participating systems (cp. figure 1(b)). Such migration is the most "natural" way of executing a distributed process – as inherited by traditional human-based workflow management: A process is described in subsequent steps which are passed from one workplace to another. Logical fragmentation by process instance migration has several advantages over physical process fragmentation:

- Process migration allows for fragmenting the responsibility to execute a process at runtime – depending on the availability of business partners or other contextual incidences. Furthermore, the granularity of fragmentation and the range of distribution can be selected on the fly by each executing participant.
- Coordination and merging of multiple process fragments are not necessary in the case of sequential execution. Global variables, scopes, errors and transactions are easier to handle, because all these aspects of the process (i.e. data and control flow) are available to all executing parties. Thus, there is less communication and coordination overhead.
- Process instance migration is applicable to modern distributed systems including mobile devices because it does not depend on a single centralized system and allows for dynamic sharing of restricted resources [3,5].

However, process migration has also some drawbacks and still includes some interesting challenges which this paper proposes to address. First, the process description needs to implement a formal or technical model to communicate the current state of the migrated process instance. To preserve interoperability, this model should not require modifying the original business process [2]. Second, an important motivation for physical process fragmentation is given by the resulting separation of process fragments. If the process is to be fragmented for privacy reasons, process migration lacks proper security mechanisms in order to protect private information carried within the process.

This paper presents an approach to enhance existing processes with non-intrusive migration metadata and an overall system architecture to support runtime process migration among cooperating process execution systems. Therefore, we analyze which information has to be attached to the process at designtime in order to execute the (logical) process fragments as it was originally intended by the designer or the initiator of the process in whole. Furthermore, we present an initial privacy mechanism to protect the migrating process instance against unwanted changes and unauthorized access. Finally, the approach presented here is applied to WS-BPEL and a respective prototype implementation, before the paper concludes with a short summary.

2 Related Work

Distributed and decentralized process execution becomes increasingly important and, consequently, many such approaches demonstrate the relevance of this research (cp. [6] for a brief overview). A first possible solution for distributing the control flow of a process is to change the service granularity. The activities which should be outsourced are wrapped, encapsulated behind a new service interface and the remaining process model is changed accordingly (e.g. [7,8]. However, such process fragmentation is mostly carried out at design time and is realized by weaving additional activities into the resulting fragments in order to realize a standard-compliant communication between them at runtime. Another general approach is to split the original process, deploy the resulting parts at the desired system and induce *choreography* between the separated processes (e.g. *CiAN* [9]). However, choreography and process fragmentation need a joint preparation phase for the physical distribution of each (sub-)process where all participating parties have to be available. Therefore, this approach is more advantageous in case of a similar recurrent execution of the same process than for spontaneous reactions to (unfrequent) ad-hoc changes.

In order to avoid the introduction and maintenance of new services, Martin et al. [2] propose a non-intrusive approach for process fragmentation and decentralized execution. Here, fragmentation is achieved by transforming the orchestration logic represented in WS-BPEL into a set of individual activities which coordinate themselves by passing tokens over shared distributed tuple-spaces. Decentralized process execution has also been considered in Mentor [10] by partitioning a process based on activity and state charts. Addressing more

dynamic environments, the approach of *MobiWork* [11] realizes mobile workflows for ad-hoc networks and is focused on the allocation of tasks to mobile participants also using process fragmentation to generate "sub-plans".

However, all presented approaches support at most dynamic allocation and assignment on the basis of a *static* fragmentation. All fragments and responsible parties are determined either at design time or once after invocation but mostly before executing the first activity of the process instance. Considering long-running processes, this flexibility may not be enough in order to also allow reactions to spontaneous contextual changes. In contrast, a *dynamically continuable* runtime segmentation implies that fragments and responsible parties are determined dynamically according to the current context and with respect to previous results and requirements of upcoming activities during the actual execution of the process instance.

A way to address such dynamic behavior is the runtime migration of entire process descriptions as introduced by Cichocki and Rusinkiewicz [12] in 1997. More recently, the framework *OSIRIS* [13] relies on passing control flow between distributed workflow engines in order to execute service compositions. Process data is kept in a distributed peer-to-peer-database system which can be accessed from each node participating in the process execution. In *Adept Distribution* [14] a similar approach supports dynamic assignment of process parts to so-called *execution servers*. A process instance can be migrated from one execution server to another and the next participant is dependent on previous activities which are able to change the participant to execute the next partition. Related to this, Atluri et al. [4] present a process partitioning algorithm which creates self-describing subprocesses allowing dynamic routing and assignment.

Process migration has also particularly been applied to the area of mobile process execution, e.g. by Montagut and Molva [5]. Their approach relies on passing control flow between distributed WS-BPEL engines and addresses security on an application level by integrating a public/private process model in order to access applications internal to mobile devices. However, such a solution represents a choreography-like approach which only uses process migration in order to hand-over control flow – and thus also has some of the aforementioned disadvantages. The last example is the *DEMAC* middleware [3] which is able to delegate process execution (in whole or in part) to other stationary or mobile process engines. Its restriction to a proprietary process description language is, however, an obstacle to migrate existing business processes and to integrate standard process engines of external parties.

Complementing existing migration approaches, we therefore introduce a more technology-independent migration model which can also be applied to existing processes. Furthermore, most migration approaches give scope for rather too much flexibility, i.e. the process instance is migrated without control or the decision about the next participant is determined by one of the execution systems - but often cannot be influenced by the process modeler or initiator. The following section presents a respective concept while considering the above-mentioned user-defined requirements for logical process fragmentation.

3 Overview of Process Instance Migration

The proposed methodology of process instance migration is depicted in figure 2. The development starts with the original modeling of the underlying business process which produces a process model, specified in an executable process description language such as WS-BPEL (step 1). Optionally in step 2, this process model can now be enhanced by a *migration metadata model* which holds all information required for migration and all user-defined requirements. In the following, process model and migration model are deployed (step 3) and can be instantiated by an application or a user (step 4). If required, parameters are passed to customize the process (i.e. normal invocation parameters) or the migration model. The latter is advantageous if the initiator should be allowed to influence non-functional aspects about the way a process is executed (e.g. if the user pays for a higher service quality, the selection of migration partners is influenced accordingly). After that, the resulting process instance is executed following the guidelines of the associated migration metadata. However, if migration metadata is omitted or migration is not supported, the unaffected process can still be deployed and executed the usual (centralized) way.

The proposed migration model and its relationship to general process elements are depicted in Figure 3. As a starting point, we assume a common minimal process model consisting of a finite number of *activities* representing the tasks to be fulfilled during process execution, and a finite number of *variables* holding the data which is used by these activities. Activities can represent a specific task (*atomic activities*) or a control flow structure as a container for other activities (*structured activities*). Furthermore, variables can be specified on process level (*global variables*) or at activity level (*local variables*). Optionally, variables can contain *initial values* which are assigned at design time.

A process description complying to these properties (e.g. XPDL or WS-BPEL) can be enhanced by migration metadata documenting the execution state of the process (*process state*) and of each activity (*activity state*), such that the progress in processing the activities is well-defined and visible for every participating

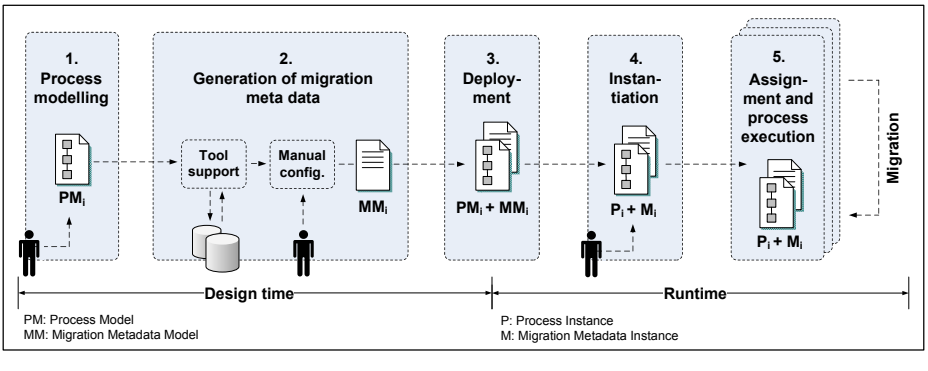


Fig. 2. Process migration: methodology

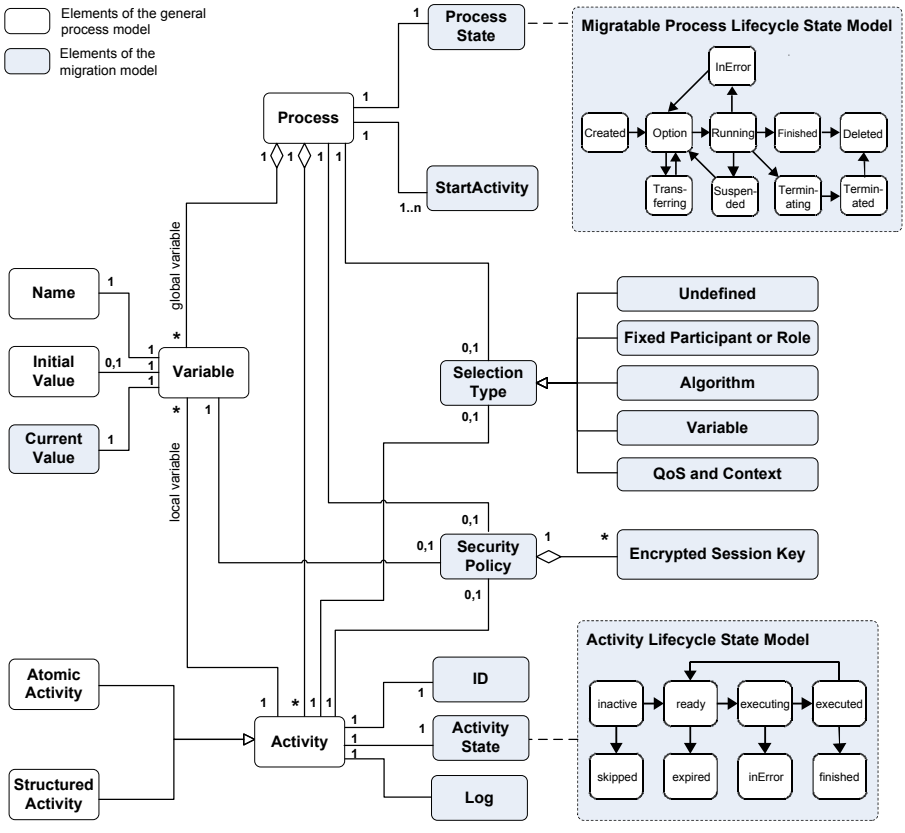


Fig. 3. Overview of the migration model

device at any time during execution. The process state can take a value from the *migratable process lifecycle model* (cp. [3]) as depicted in the upper right corner of Figure 3. In addition to that, a set of activities can be referenced as *startactivities* to mark the first activity to be executed after process migration. The model allows for multiple startactivities in case the order in which the activities have to be executed is irrelevant or the activities should be processed in parallel. Furthermore, the states of the variables have to be documented, i.e. their *current values* have to be attached to the migration data.

Up to this point, basic migration metadata can be generated automatically, i.e. by setting the process to the state *created* and all activities to *inactive* (cp. first part of step 2 in figure 2). If variables have been specified with an initial value, the given value is set as the current value of the variable. However, the process modeler or the actual initiator often wants to customize the way the distributed process is executed. The *selection type* determines which strategy is used to assign an activity to a specific process engine. If the selection type is *undefined* (default) the process engine which is currently working on the process instance decides about further migrations, e.g. to shift processes to other engines

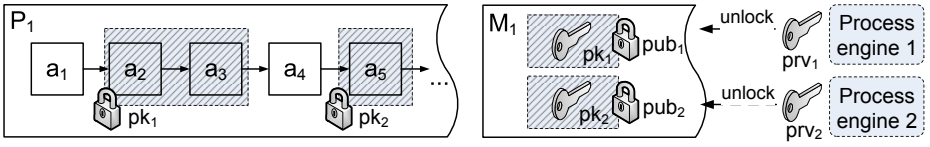


Fig. 4. Process encryption and key distribution

which operate at a better performance. In contrast, the type *fixed participant or role* determines that a specific executing entity (e.g. a human or a concrete process engine) or a subject of a defined group of such entities has to execute the process or a specified set of activities. More dynamically, as proposed by [14], the next participant can also be picked from a *variable* within the process description itself. If no such entities should be specified, but the participant should be selected as a result of a computation (e.g. picking the process engine which can execute as much of the process as possible), the respective *algorithm* is referenced. Finally, the selection can be based on specific *quality of service or context* requirements such as current workload or geographical location.

To prevent privacy and security threats, the access to process data can be restricted to specified subjects or roles, as e.g. determined in the above mentioned selection types *fixed participant or role*. Figure 4 shows the general idea of "masking" critical parts of a process description in order to ensure that only dedicated participants can execute sensitive activities and access corresponding data. The approach assumes a basic cryptographic key infrastructure, such as PKI (Public Key Infrastructure) or subject-related shared keys. However, most process description languages (such as also WS-BPEL) allow for the definition of global variables which can be referenced in several activities – and thus might belong to more than one participant. In consequence, these parts cannot be directly encrypted with the personal key of the authorized subjects. Alternatively, the encryption of the different parts of the process (i.e. activities, variables or even the whole process) uses different *session keys* which are only used once. A corresponding *security policy* of such an element therefore contains a number of symmetric keys (e.g. pk_1 and pk_2 in Figure 4). The procedure of key distribution is based on a concept which is derived from broadcast encryption [15] where the same encrypted content is sent to all receiving parties without the need for two-way authentication or authorization. In the approach presented here, the keys necessary for decryption are sent together with the protected content. In case of an existing PKI the entries are encrypted with the public key pub_i of the appropriate subject (cp. Figure 4) and can be unlocked with the private key prv_i. Accordingly, an entry for each authorized subject is created and added to the migration meta data of the protected process element. As the result of this step, only the legitimate receiver is able to obtain the keys and decrypt the content and even encrypted global variables can be accessed by different authorized subjects using the same session key [16]. Using MACs (Message Authentication Codes) for each security-related process part, the process initiator is optionally able to also ensure the integrity of the process description. Neither

an additionally interaction between the process initiator and the subjects nor an authentication is needed. As a positive side-effect, the use of unique session keys also increases the resistance of the cryptographic approach to attacks.

4 Migration of WS-BPEL Process Instances

WS-BPEL is a block-structured XML-based process description language which allows composing web services. According to the WS-BPEL 2.0 specification by OASIS [1] it is essentially comprised of two kinds of activities: Basic activities for web service interaction (*invoke, receive, reply*), basic control flow activities (*empty, wait, exit, throw, rethrow*) and activities for data manipulation (*assign*). Structured activities are used to compose the basic activities and define control-flow dependencies between them (*sequence, if then else, pick, flow, while, repeat until, for each*). Based on this characterization, the activities have been assigned to the elements of the general model in figure 3 and migration possibilities have been analyzed using a prototypical implementation of a software component responsible for performing the migrations.

Considering atomic activities, it shows that WS-BPEL has a very interactive character which makes the distribution of the control flow logic (both for migration and for fragmentation) difficult. The *invoke* activity initiates the invocation of a web service which is specified within the process description (or references associated parts such as WSDL files) in either an abstract or a specific way. Thus, migration of a process containing an *invoke* activity is not only possible, but even advantageous if the required service is not reachable from the current system. In case of a synchronous service call (*request-response* pattern) the receipt of the response message is part of the atomic activity. In case of asynchronous messaging, sending an associated *reply* subsequent to a migration is also not critical as the required information about the receiver (e.g. its physical address) is logged. Nevertheless, receiving a reply (*receive*) requires the specification of a specific participant because the sender of the reply has to know where to send the message. In this case, the flexibility of distribution is limited.

The assignment of a variable (*assign*) is not a problem as the current value is stored within the migration metadata. The same is true for *wait*, *empty* and *exit* activities as these have a rather simple behavior. Notifications about faults are also uncritical as in case of process migration all the relevant information for fault handling (i.e. *scopes*, *fault handler*, *compensation handler*) are available to the executing party. If required, the occurrence of faults can also be documented in the log, e.g. if the control flow logic has to return to the failed activity when fault handling is finished. In contrast to process fragmentation, coordination is not required in order to notify other process fragments.

Structured activities such as *sequence* or *while* do not have to be finished in order to allow the migration of the process instance. This is another advantage over physical process fragmentation where e.g. loops often have to be distributed as a whole: By storing the current value relevant for the evaluation of the loop condition in the migration data, migration is even possible within iterations.

If the condition has to be evaluated only once (such as in the case of *if then else*) the selected branch is determined by the process's startactivity. In case of process fragmentation, the execution of resulting *dead path eliminations* requires further coordination if process fragments are distributed physically. In case of process migration, the dead path can be processed automatically by setting all upcoming activities (until the next join condition) to the *skipped* state. As this information is hold in the migration document, this does generally not involve communication with other participants.

The *pick* activity waits for the occurrence of an event from a set of events and then executes the activity associated with that event. If the process is fragmented physically, this is a problematic issue. Either all the necessary data has to be replicated (i.e. all event/reaction pairs) or the events have to be fragmented as well. If the reaction to an event affects other fragments, additional coordination is necessary. In case of process migration, this is not a problem as the whole spectrum of possible events and reactions is available to the responsible participant. If, furthermore, other activities are temporarily suspended because of the event, the activity states indicate where the execution must be continued. However, each process participant has to subscribe to each required event as long as it is responsible for the execution of the process instance. Thus, during migration time, there is a remaining risk that some events may be not be noticed.

The *flow* activity contains activities which should be processed in parallel. As long as the process is migrated to exactly one participant, migration within the execution of a flow is uncritical as the states of each included activity are well-defined. Nevertheless, the process cannot be transferred until all atomic activities have reached a stable state and thus may have to wait for long-running activities to be finished. Since the execution of parallel paths on a single machine cannot be considered as "real parallelism", a copy of the (entire) process can be distributed to different participants which are each responsible for the execution of one of the parallel paths. In order to synchronize parallel paths, there has to be a defined meeting point. In consequence, distributed parallel execution needs advanced coordination mechanisms for both migration and fragmentation. However, using replication instead of fragmentation allows for a local detection of shared variables and thus avoids unnecessary synchronizations.

As a drawback for process migration, privacy can only be realized by artificially masking private process parts, whereas physical fragmentation of the process makes such mechanisms unnecessary. In consequence, the effort for developing migratable processes containing private parts is a little higher – and process modelers must be aware of the fact that applying such policies reduces the number of potential migration partners and thus again may restrict flexibility. Nevertheless, even within user-defined limitations, process migration allows for more flexibility than physical process fragmentation – and especially long-running distributed process instances benefit from the possibility to adapt the remaining unrestricted execution of control flow to changing conditions.

5 Conclusion

This paper focuses on distributed process execution involving multiple engines in order to increase flexibility and to improve reactions to ad-hoc context changes. As an alternative to physical process fragmentation, a concept for realizing logical process fragmentation on the basis of process migration has been presented. A prototype system covering the proposed system architecture for XPDL and WS-BPEL processes shows basic applicability of the proposed concepts. Compared to physical fragmentation, process migration provides more flexibility by allowing to distribute running process instances at runtime while respecting the guidelines of the process modeler. On the other hand, privacy and security-related issues have to be considered explicitly as also addressed in this paper.

References

1. OASIS: Web Services Business Process Execution Language Version 2.0. Technical report, OASIS (2007)
2. Martin, D., Wutke, D., Leymann, F.: A Novel Approach to Decentralized Workflow Enactment. In: Enterprise Distributed Object Computing, pp. 127–136. IEEE, Los Alamitos (2008)
3. Zaplata, S., Kunze, C.P., Lamersdorf, W.: Context-based Cooperation in Mobile Business Environments: Managing the Distributed Execution of Mobile Processes. In: Business and Information Systems Engineering (BISE), vol. 2009(4) (October 2009)
4. Atluri, V., et al.: A Decentralized Execution Model for Inter-organizational Workflows. Distrib. Parallel Databases 22(1), 55–83 (2007)
5. Montagut, F., Molva, R.: Enabling Pervasive Execution of Workflows. In: Collaborative Computing: Networking, Applications and Worksharing. IEEE, Los Alamitos (2005)
6. Jablonski, S., et al.: A Comprehensive Investigation of Distribution in the Context of Workflow Management. In: ICPADS 2001, pp. 187–192 (2001)
7. Khalaf, R., Leymann, F.: A Role-based Decomposition of Business Processes using BPEL. In: IEEE International Conference on Web Services, pp. 770–780. IEEE, Los Alamitos (2006)
8. Baresi, L., Maurino, A., Modafferi, S.: Towards Distributed BPEL Orchestrations. ECEASST 3 (2006)
9. Sen, R., Roman, G.C., Gill, C.D.: CiAN: A Workflow Engine for MANETs. In: Lea, D., Zavattaro, G. (eds.) COORDINATION 2008. LNCS, vol. 5052, pp. 280–295. Springer, Heidelberg (2008)
10. Muth, P., et al.: From centralized workflow specification to distributed workflow execution. J. Intell. Inf. Syst. 10(2), 159–184 (1998)
11. Hackmann, G., Sen, R., Haitjema, M., Roman, G.C., Gill, C.: MobiWork: Mobile Workflow for MANETs. Technical report, Washington University (2006)
12. Cichocki, A., Rusinkiewicz, M.: Migrating Workflows. In: Advances in Workflow Management Systems and Interoperability, NATO, pp. 311–326 (1997)

13. Schuler, C., Weber, R., Schuldt, H., Schek, H.J.: Scalable Peer-to-Peer Process Management - The OSIRIS Approach. In: ICWS, pp. 26–34 (2004)
14. Bauer, T., Dadam, P.: Efficient Distributed Workflow Management Based on Variable Server Assignments. In: Wangler, B., Bergman, L.D. (eds.) CAiSE 2000. LNCS, vol. 1789, pp. 94–109. Springer, Heidelberg (2000)
15. Lotspiech, J., Nusser, S., Pestoni, F.: Broadcast Encryption's Bright Future. Computer 35(8), 57–63 (2002)
16. Bertino, E., Castano, S., Ferrari, E.: Securing XML documents with Author-X. IEEE Internet Computing 5(3), 21–31 (2001)