In: Proc. of the EWGLP 2000 "1st European Workshop on Computer-based Support for Clinical Guidelines and Protocols". Leipzig, Germany, November 2000.

Transactional Execution Guarantees for Data–Intensive Processes in Medical Information Systems *

Christoph Schuler Heiko Schuldt Hans–Jörg Schek
Institute of Information Systems
Swiss Federal Institute of Technology (ETH)
ETH Zentrum, 8092 Zürich, Switzerland
Email: {schuler,schuldt,schek}@inf.ethz.ch

Abstract. Medical information systems are characterized by the existence of various distributed and heterogeneous software systems, each of them specialized for certain domains. In this environment, interoperation between systems is crucial since, in general, large amounts of data such as multimedia objects have to be transferred. Mostly, this is achieved by some communication server which supports bilateral data transfer. However, communication servers do neither allow to have a global view on dependencies between all systems nor do they provide for any transactional execution guarantees such as failure handling and/or alternative executions that include more than two systems. This, in turn, can be solved by encompassing global dependencies into processes and by exploiting systems that execute processes under transactional control. But these systems are usually not made for the support of data-intensive applications. Hence, a combination of transactional process support and communication servers is required such that the advantages of both systems can be brought together.

In this paper, we present an approach to support data-intensive processes for which transactional execution guarantees are provided. We have implemented a prototype system based on the transactional process management system WISE, developed at ETH, and the commercial communication server Cloverleaf. By plugging Cloverleaf as a dedicated component to the WISE system, we are able to execute transactional processes in medical information systems while, at the same time, supporting the standard communication server functionality for data transfer purposes.

1 Motivation

1.1 Introduction

Medical information systems are characterized by large contents of data, stemming from various sources, i.e., distributed, autonomous, and heterogeneous application systems. Patient records, for instance, which have to exist for each patient who has ever been treated in a hospital, contain all data from previous treatments and examinations. The latter encompass, in general, large volumes of multimedia data objects like radiographs

*Part of this work has been funded by the Swiss National Science Foundation under the project A META MODEL FOR MODELING AND MANAGEMENT OF LARGE SCALED INFORMATION SYSTEMS

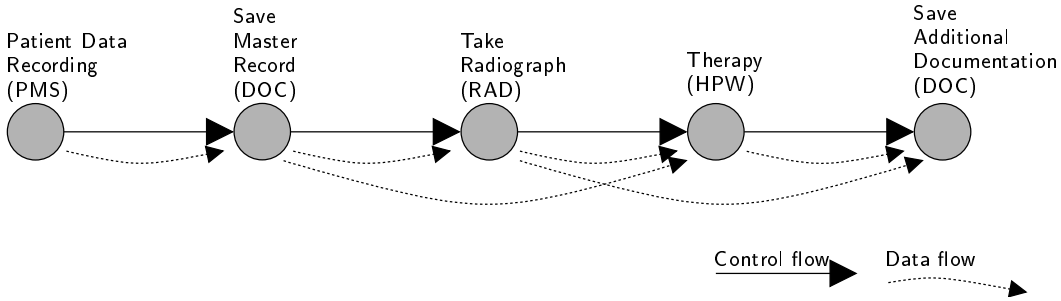

Figure 1: *Sample Process: Medical Treatment*

or spoken reports. Although these records are stored in a centralized database (patient management system), data is produced by external applications such as, for instance, X-ray systems, and has to be transferred to the patient management system for persistent storage. In general, each department in a medical domain has its own specialized tools and programs. In order to exchange data between departments, interoperability between these applications is required. Hence, data transfer might also require a peer-to-peer communication between these applications rather than transferring data only to the patient management system. In the presence of large images or multimedia objects, this may lead to high data traffic.

Such distributed applications spanning multiple independent tools and systems are usually specified by means of processes [16]. Processes can be divided into a set of self-contained units, i.e., activities, which correspond to invocations of services provided by existing tools or programs, that take place one after the other. This chain of activities can be very complex, since there are usually several independent departments and a number of distributed tasks involved. These processes, as they can be found in real-world applications in medical domains, can be characterized as complex, long–running, and labour–intensive. In particular, they require certain coordination among their constituent steps, mainly in terms of appropriate failure handling strategies and of strategies for synchronizing parallel processes.

In various other environments (e.g., banking business), global coordination of processes is mostly solved by distributed transactions orchestrated by TP–monitors on top of database systems. But these products limit the autonomy of applications and impose very strong restrictions which are unrealistic in medical information systems. Hence, processes in medical information systems require more flexible transaction models including sophisticated failure handling strategies while avoiding the limitations imposed by traditional transaction models.

1.2 A Sample Process

In the following, we introduce a sample process (shown in Figure 1) to illustrate a set of typical dependencies which exist between the diverse steps that have to be taken for a medical treatment. In practice, this kind of process may be much more complex. In here, we use this simplification in order to focus on the features which are essential to our approach and to motivate the major aspects described in this paper: Assume that a patient, Bob, is taken to a hospital where first his name and address are recordedin some patient management system (PMS). This data is sent to a central database, the documentation system (DOC), where a patient record for Bob is created or, in case this record already exists, data of previous treatments is retrieved. In the latter

case, eventual changes, e.g., of Bob's address, may lead to an additional update of the patient record. After reception, Bob is sent to the radiology department where X-ray examinations are performed and a radiograph is taken (RAD). However, Bob's patient record has to be sent to the radiology department prior to the examination. The same is true for the therapy room where the next step of his treatment takes place, supported by a Health Professional Workstation (HPW); yet, the patient record is also required there. In addition, the radiograph has to be sent to the therapy room after the X-ray examination. Based on the information extracted from the radiograph, the responsible physician decides that a shot is to be dispensed. At the end of the treatment, all data that is related to the therapy is stored in the central database (DOC), as part of Bobs patient record, including the radiograph and a protocol of the treatment written by the responsible physicist.

2 Communication Servers for Data–Intensive Applications

Communication servers (e.g., Cloverleaf [2] or e*Gate [1]) are key components in current medical information systems since they are tailored to the problems that frequently occur in these environments. In short, the functionality of these systems can be characterized by the following two features. First, they support the application programming interfaces (APIs) provided by the most relevant application systems that are used in medical domains in order to allow them to interoperate. Second, they support the transformation of data between various standards in which applications publish their data.

In the context of multimedia documents (such as radiographs or spoken reports of medical treatments) which are part of patient records, large volumes of data must be transferred between applications. Hospital application systems normally publish their changes by message types such that the latter can be exploited by consumers which rely on the information wrapped within the message. These messages are sent whenever an interesting event occurs or when another system requests some information. Messages can be of any size, leading from a short notification in plain ASCII to the submission of a radiograph.

To exchange messages between different medical application systems, a set of standardized messages is used. However, since no commonly agreed standard for data formats exists (there are actually different standards but applications, in general, only support a subset of them; even worse, certain systems only support proprietary formats), a converter in between each pair of applications that aim at exchanging data is required. To this end, the message has to be translated into the corresponding message format of the destination system and to be forwarded to that system in the corresponding protocol type. This task is commonly performed by a communication server. The routing takes place on the basis of special information in the message (e.g., the message type) and by pre–defined routing tables which contain meta information of the transfer indicating how to handle certain types of data transfers, and where to route data. Hence, no application has to maintain a list of systems interested in this message type (consumers); they have rather to send the notifications and requests to the communication server. These kinds of interactions are also referred to as publish–and–subscribe [18] techniques. The server redirects requests to the desired system and returns acknowledgments. In general, each data transfer via the communication server is encapsulated in two separate transactions: the first one considers the message transfer from the originating application to the communication server while the second one

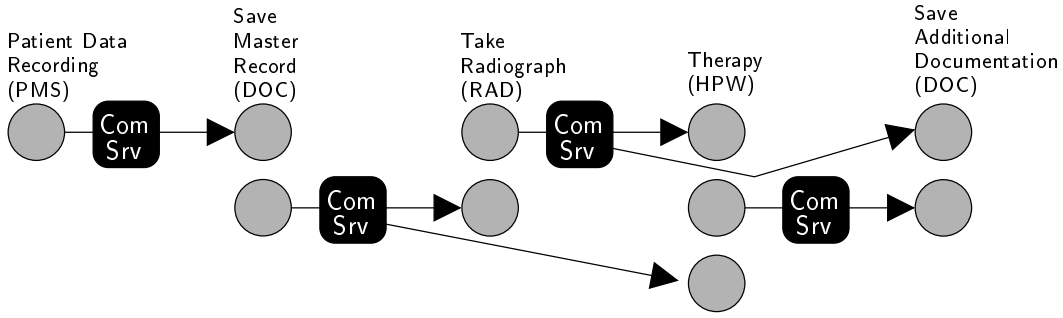

Figure 2: *A communication server (ComSrv) routes messages to the next application step*

consists of the message delivery from the communication server to the consuming application. Although message transfer is, in general, split in independent transactions, the delivery of a message is guaranteed by the communication server (e.g., by repeatedly contacting the consuming application if it is temporarily unavailable). However, in case consistency is essential and synchronous messaging is required, the transfer can be encapsulated within one single transaction such that the originating application has to wait until an acknowledgement of successful transfer arrives from the consuming application; this notification is also transferred via the intermediary communication server.

Since input data for certain applications may stem from different sources, they may not be available at the same time in the same transaction. In this case, the destination system has to gather all information and store them for future use (i.e., patient data and radiograph at the 'Therapy' step in Figure 2).

Coming back to the initial example depicted in Figure 2, the application system (DOC) on 'Save Master Record' cannot wait synchronously for the 'Save Additional Documentation' step, because there are several other actions in between these two steps. According to the philosophy of communication servers, the transfer of the radiograph from the X-ray (RAD) system to the patient database is actually decomposed into two independent transactions: (i) data transfer from the X-ray examination to the communication server, and (ii) the subsequent transfer from the communication server to patient database.

Since all functionality is distributed to heterogeneous applications, there is no global view on the process. And, even worse, no global coordination is possible. Each application publishes certain events and generates messages without being aware of the steps that have to be taken subsequently and the applications that are involved in these steps. Even the communication server has no global view on dependencies but manages only the bilateral transfer of data. Hence, the problem on how independent application services and/or transactions can be glued together while exploiting information on global dependencies remains unsolved.

3 Providing Execution Guarantees: From TP–Monitors to Transactional Process Management

3.1 Distributed Transactions in TP–Monitors

Transaction Processing Monitors (TP-Monitors) [12, 8] implement distributed transactions on top of heterogeneous, distributed database management systems. They are intended to bring together many different database systems so as to operate as one

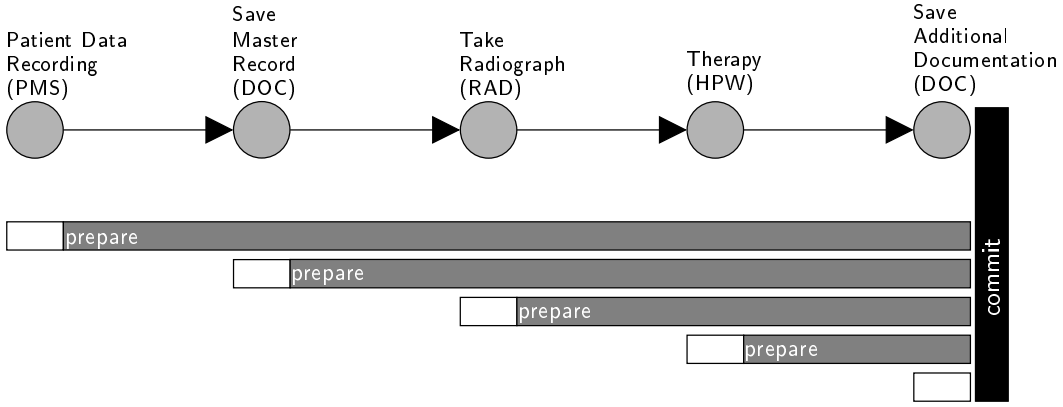

Figure 3: *Distributed transactions realized by Two Phase Commit*

single logical database. TP–Monitors are thus responsible to guarantee the overall consistency of distributed systems. TP–Monitors are typically realized in a three tier architecture. They use an intermediate layer (middleware) which not only encapsulates the underlying systems but also transparently hides their heterogenity by exploiting a common protocol, two phase commit (2PC) [12, 7], which has to be supported by all component systems. One of the most important achievements of TP-Monitors is transparent transaction control. The client starts a transaction, performs some actions and commits the transaction as if the whole transaction would have been executed in one single database. Our sample process can be realized as a distributed transaction (as shown in Figure 3). The global transaction consists of five distributed subtransactions, each of them corresponding to an activity of the process depicted in Figure 1. All changes in the underlying applications are executed within the same scope of atomicity. This can be realized, using a two phase commit protocol, by leaving all subtransactions in a prepare–to–commit state until commit of the global transaction is reached. Whenever an error occurs, the whole transaction can be rolled back at any point in time prior the global commit. To achieve this, all resources have to be held such that attached systems are blocked until the global transaction is terminated, i.e. no component systems is allowed to unilaterally decide to abort or commit during the prepared phase.

According to the distributed transaction model, all subtransactions belong to the same logical global transaction. If one system rolls back, all other system have to roll back as well. Autonomy of the component systems is no longer provided. It is not possible for a single system to react in a different way — after having responded positively to the commit vote request — other than imposed by a global coordinator. If a global transaction is not committed and when the local subtransactions are in the prepared state (i.e., they have been executed completely and wait for a global commit decision), yet local access to the same data is blocked. This effect is essentially undesirable because autonomy is an important issue in environments like medical information systems. For this reason, the strong requirements imposed by coordination based on distributed transactions should be relaxed.

3.2 Non–Blocking distributed transactions

An option to prevent the drawback that application systems be blocked is to allow each local subtransaction to commit right after their execution as defined by the notion of open nested transactions [24]. But then, the top–level transaction can no longer be

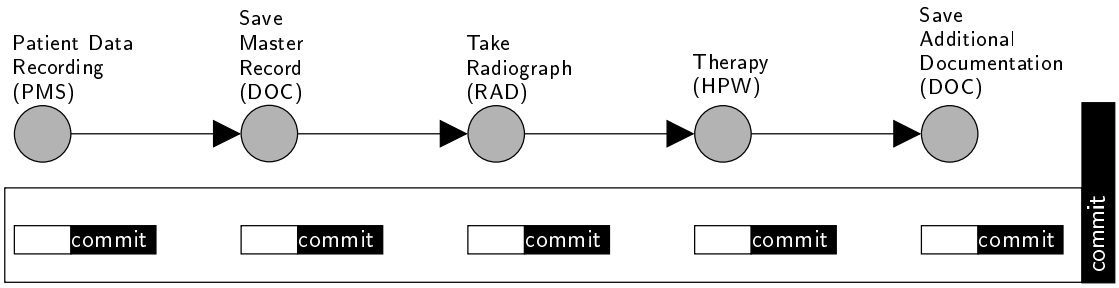

Figure 4: *Distributed transaction following the open nested paradigm*

rolled back automatically if one subtransaction fails, since some subtransactions may have already committed. Yet, a compensating subtransaction has to be provided for each regular subtransaction such that failures can be handled by applying compensation in reverse order (semantic atomicity [10]). In general, designing a compensating subtransaction may be a difficult task, since it essentially requires dedicated information about the concrete application and about the semantics of the corresponding regular transaction. Even worse, some application services, once committed, cannot be undone because of their irrevocable nature (e.g., an injection). Other services may, in principle, be compensatable albeit this might be very expensive such that their compensation is not desired in order to avoid that costly steps are discarded.

Figure 4 shows our sample process realized by open nested transactions. Due to its Complexity the 'Take Radiograph' activity should not compensated, i.e., the radiograph has to be kept for further treatments, independent whether it is currently needed or not. Assuming an injection takes places during the 'Therapy' activity. Then, it must, however, not be compensated because of the irreversible nature of an injection. An alternative strategy that considers additional treatment after a shot has been given must be applied for failure handling purposes. Hence, open nested transactions, although loosening the restrictions imposed by distributed transactions, still do not meet the requirements that can be found in medical information processes. Thus, further extensions are needed so as to add more flexibility to the transaction model, thereby allowing further improvements and generalizations.

3.3 Transactional Process Management

Classical transaction management as implemented in conventional databases follows an all–or–nothing semantics of atomicity. If a transaction cannot be terminated successfully, the whole transaction is rolled back (in the case of distributed TP–Monitor transactions), or all committed subtransactions are compensated (according to the open nested transaction paradigm). For several reasons, a full rollback can not be performed or is, in certain cases, to be avoided. Therefore, we generalize the dependencies that exist within global transactions under the notion of *process*. In short, a process which consists of a set of *activities* as basic units of execution allows for more flexibility by (i) considering activities as invocations of arbitrary application services and by (ii) considering more sophisticated failure handling strategies that are already part of the model.

Some activities of a process are long running tasks and are too expensive to be undone and restarted. Activities like injections are irreversible and cannot be undone

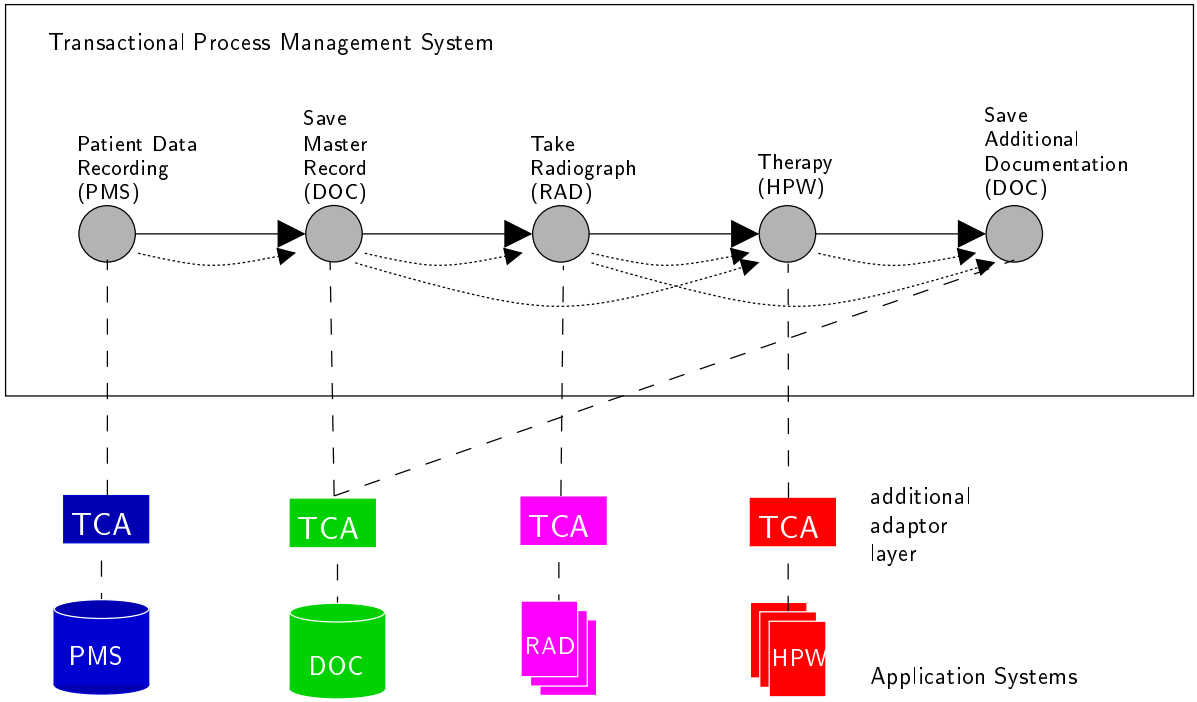

Figure 5: *Transactional Process Management on top of arbitrary Application Systems*

at all. After such an activity is successfully terminated, the associated process can only terminate successfully in forward direction, but no longer by performing backward failure recovery. To this end, first a proper classification of the termination properties of single activities is required before the characteristics of the associated processes can be addressed. In the activity model of transactional process management [20], borrowed from the flexible transaction model [15, 27], there different classes of activities are distinguished with respect to their termination characteristics:

- A **compensatable** activity can be undone by a semantical rollback of effects of the activity. To this end, an appropriate compensating activity has to be available.
- A **pivot** activity cannot be compensated.
- **Retriable** activities are guaranteed to succeed after a finite number of invocations. This means that each possible failure that occurs during execution is of a temporary nature only.

In order to cope with these different termination properties, a process is not only characterized by a regular execution order imposed between its activities but also by a so-called precedence order which specifies alternative executions that are applied in the case of failures. By these alternatives, a process can define several multiple correct executions between which priorities are implicitly assigned by the preference order. Each of these execution paths, when executed completely and correctly, must leave the global system in a consistent state, independent of the failure of single activities. This is the case when failures can either be captured by a repeated invocation of the failed activity (in case it is retriable) or if they lead to an alternative execution, i.e., execution is switched to another possible path. Based on the termination properties of all activities and the different orders imposed between them, a validation is possible

analyzing whether processes are correctly defined or not. In short, a single process, even in presence of multiple pivot activities, is correct (it supports *guaranteed termination*) when each possible failure can be handled correctly and leads to a correct final state. Hence, the inherent guaranteed termination property of a process is a generalization of the all-or-nothing semantics of atomicity since it requires that one out of several correct outcomes of a process execution is reached. Based on the inherent correctness of single processes, transactional process management additionally focuses on the correct concurrent and fault–tolerant execution of processes by generalizing and applying the unified theory of concurrency control and recovery [5, 22] for processes, thereby providing a criterion that simultaneously accounts for atomicity and isolation.

Activities of transactional processes do not necessarily have to correspond to database transactions but can be arbitrary service invocations in some underlying application system. However, the provision of execution guarantees at process level requires certain properties to be present for each of these services (e.g., a compensation service must exist in case an activity is considered as compensatable). In general, since we have to deal with arbitrary black-box applications, these properties might not be provided. To this end, an additional software layer has to be established between the transactional process management system and the underlying applications consisting of a transactional coordination agent (TCA) [21] acting as advanced application-specific wrapper tailored to each of these systems (depicted in Figure 5). Aside of exploiting the API of the underlying application, each TCA has to implement the functionality required by transactional process management in case this is not directly provided by the underlying application. In terms of compensation, for instance, a TCA can rely on information extracted from log files in order to determine how the effects of a service can be undone a posteriori.

4 Combining Transactional Process Management and Communication Servers

Transactional process management systems are well suited to control the execution of complex business processes. Communication servers are designed for data–intensive transfers and message conversions. In order to extend the functionality of both systems and in order to meet the requirements imposed by real-world medical information processes, we combine the advantages of these systems such that complex dependencies can be handled in a correct way even in the presence of concurrency and failures while, at the same time supporting data-intensive information exchange between the underlying applications. In terms of our sample process, a radiograph is sent from the 'X-Ray Examination' to the 'Therapy' step of the process (see Figure 6) by using communication server functionality for format conversion and data transfer, while the execution of the process is under the control of the transactional process management system. In particular, the latter determines when activities have to be started and guarantees correct execution, e.g., applies failure handling if necessary. In what follows, we refer to a system providing this dual functionality as TxProc/IDF system (transactional process management for intensive data flow).

To realize a TxProc/IDF system, the invocation of activities and the data transfer from/to the associated applications has to be decoupled. In transactional process management systems, data flow between activities is realized via input and output containers. These containers are shipped, via the corresponding TCA, to the underlying application. Since the goal is to avoid large data objects to be passed via the process

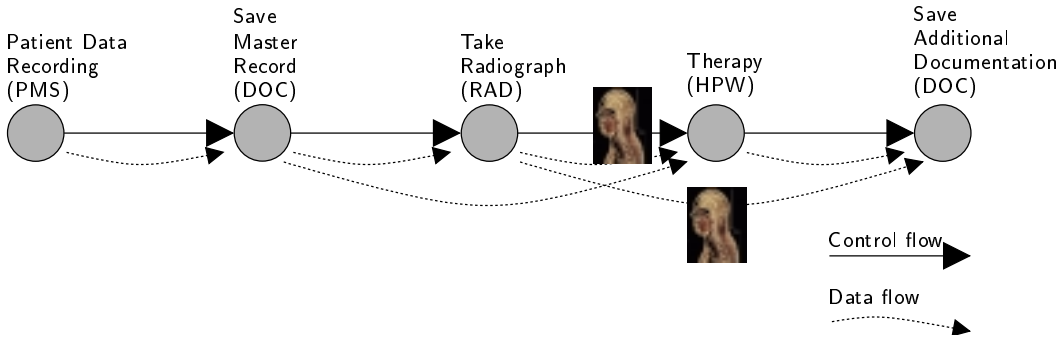

Figure 6: *Separation of control flow and data flow for large data objects (i.e., radiograph)*

management system, there must be some place where data flow can be redirected. After the completion of activities, large data objects should be sent to a communication server, while control has to be returned to the transactional process management system. This task of redirecting data flow will be performed by the TCA of the respective application that generates data–intensive parameter transfer.

Since the process management system performs navigation based on the global process, it has to make sure that all parameters sent through the communication server are available to an activity at the time it is invoked. Therefore, each message sent to the communication server has to have a corresponding reference which holds all meta–information. Instead of the data object itself, this reference is part of the data flow at process level. Hence, two parts of a large data object, the data itself and its reference, must be characterized by a unique ID in order to be able to bring them together, in addition, the transfer of both independent parts has to be synchronized.

An essential feature of a transactional process management system is to keep the state of processes persistently, thus allowing processes to recover in forward direction after system failures. Hence, also the parameters of process activities have be held in persistent storage. While this is easy to achieve for data that is transferred via the process management system, this is no longer the case for data objects that are shipped via a communication server and thus, never reach the transactional process management system. But if the reference parameter sent to the process management system is stored persistently in the run-time process database, while the communication server persistently stores the data object that is part of the message sent to the communication server, even external data can be recovered after a system failure by the unique identifier of both parts.

Normally, the communication server keeps all messages persistent while transferring them. However, after successful delivery, the message will be deleted. Since transactional process management requires the availability of all parameters at least for the lifetime of the associated process, the communication server functionality has to be extended in order to keep the messages until the end of the process. Alternatively, an additional external temporary persistent storage can be exploited for this purpose.

5 Implementation

Within the TxProc/IDF project, we have implemented the aforementioned concepts based on the transactional process management system Wise [4] and the communication server Cloverleaf [2]. Before we go into the details of the combination of both systems, we first take a closer look at the Wise system.

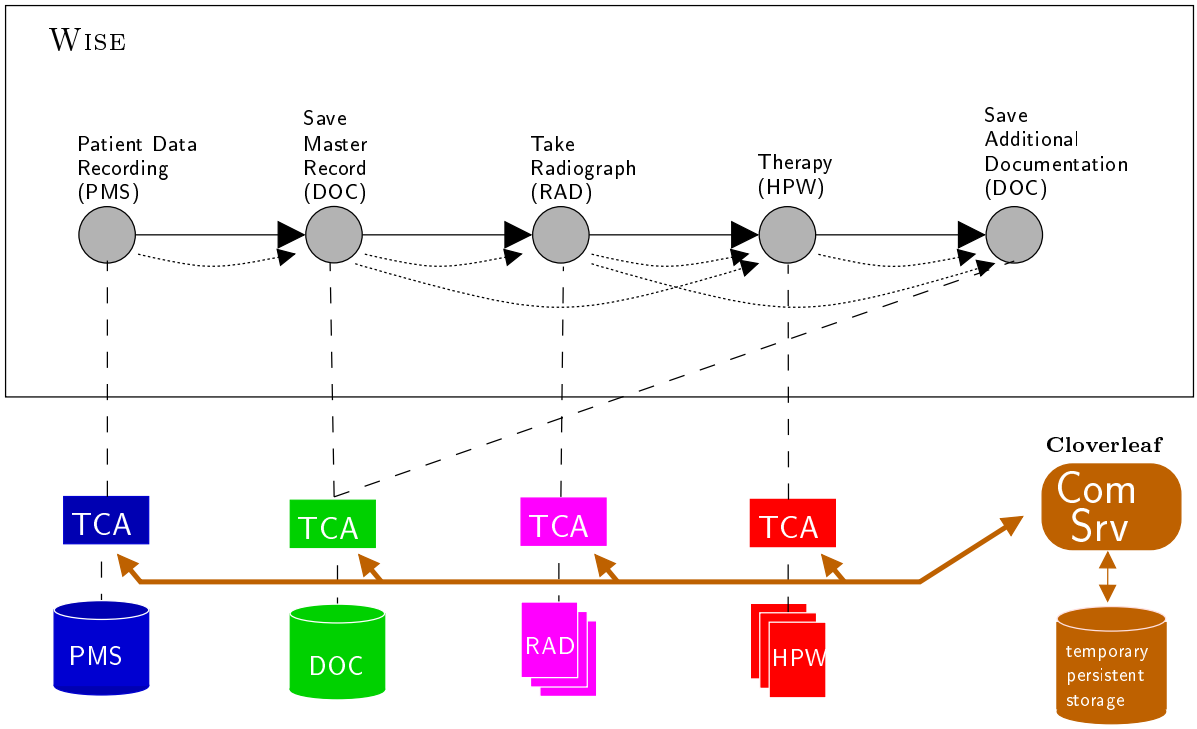

Figure 7: *Combination of the* WISE *system and Cloverleaf*

5.1 Architecture of the Transactional Process Management System WISE

WISE is a process management system that supports transactional processes. Its modular architecture consists of a small kernel that provides the functionality required for navigating through transactional processes, while specialized adapters (TCAs) can be plugged in so as to support various subsystems.

Processes are stored and managed in the kernel of the WISE system, while activities correspond to service invocations in underlying subsystems. On top of the WISE kernel, a scheduling component has been implemented which controls the access to shared resources in the presence of concurrent processes. Moreover, the basic functionality of the WISE system is extended by diverse components, for instance for the monitoring of process states. In order to support the specification of processes, the commercial process specification and simulation tool IvyFrame [3] has been extended to export processes modeled in this tools in a format that can be executed by the WISE system.

TCAs that are exploited for application integration purposes encapsulate arbitrary systems in that they provide a standardized interface to realize a seamless integration. Communication takes place only between TCAs and the WISE system; no data is transferred directly between applications or between two TCAs.

5.2 Integration of Communication Servers

The description of a TxProc/IDF process contains all information on how to invoke activities in the context of their subsystem. Besides this information, the process specification contains import and export parameters for each activity of a process. These parameters are then shipped to/from the corresponding applications via input or output containers, respectively.

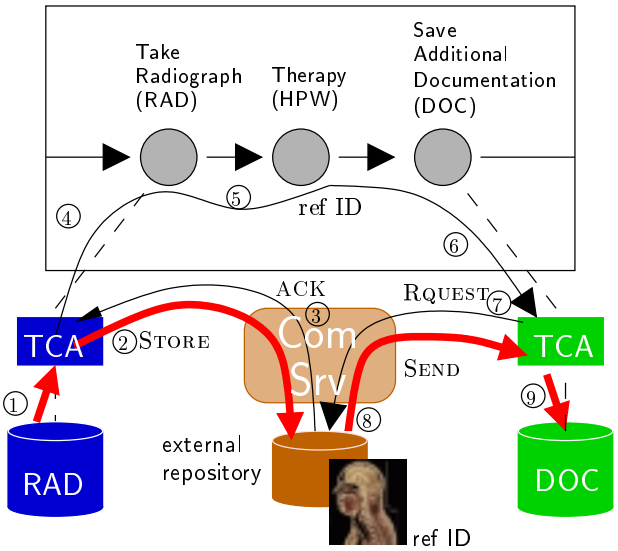

Figure 8: *Persistent storage of large data object in an external temporary storage*

In addition to standard data types, references to external data objects are also supported. These parameters refer objects which are shipped via communication servers and which are stored at some temporary persistent storage (shown in Figure 8). To support externally stored parameters, the WISE system has to be extended in several directions.

The TCA of a supported system has to split large output parameters in the presence of large data objects into a Cloverleaf compatible message also comprising the data itself and a WISE reference parameter. These two objects have to be given the same ID. Then, it is possible to subsequently retrieve the object based on the reference parameter.

In our sample process, the transfer of a radiograph from the radiography (RAD) to the documentation system (DOC) is required. Figure 8 shows this part of the process. After the radiograph is available in the RAD system (1), the corresponding TCA takes control and sends the radiograph to Cloverleaf (2). Based on the internal routing tables, the radiograph is forwarded to the temporary persistent storage; after it is stored, an acknowledgement (3) is sent back to the TCA. The external storage can be located anywhere in a network. An arbitrary database system supporting binary large objects (BLOBs) can be used for this purpose. At this point, it is guaranteed that the radiograph is stored persistently and such that the reference parameter is valid. Now, the TCA is able to confirm the successful execution of the activity and return control to the WISE system (4). As part of the data flow, the reference which is part of the output container of the X–Ray Examination (5) will become part of the input container of Documentation (6). Then, WISE navigates through the process and starts the 'Save Additional Documentation' activity. The DOC–system TCA parses the input parameters and finds the reference parameter which points on the radiograph. It sends a REQUEST message to Cloverleaf that routes the message to the external storage in order to retrieve the object (7). By responding to the request message, Cloverleaf directly forwards the radiograph object to the DOC–system (8).

After reception of the data, the TCA of the DOC–system replaces the reference parameter in the invocation string it has received from WISE by the actual data object and starts the service associated with the 'Save Additional Documentation' activity (9). Since the radiograph is stored persistently, it can be requested at any point in time by a TCA, that is, it can serve as input parameter for any subsequent activity. After

successful termination of the process, the data object temporarily stored is no longer needed. Therefore, it can be deleted.

While the transfer of externalized data objects via the communication server and the external temporary persistent storage is orthogonal to the functionality of the WISE system, the full range of flexibility and transactional execution guarantees at process level can be provided.

6 Related Work

The communication server architecture to integrate heterogeneous systems in hospital environment is based on the application integration model developed in Heidelberg [26].

The idea of communication server is strongly related to persistent queuing functionality as it is used by Publish–and–Subscribe [18] systems. These systems are also characterized by decoupling producers and consumers of data where the former do not have to have any information about a system requesting this data.

The architecture of the WISE system is based on a centralized process management system whose kernel is responsible for the navigation of processes. Other approaches rather address a distributed architecture. The ADEPT$_{distribution}$ [6] approach, for instance, handles variable server assignment in workflow management in order to minimize the communication load.

In the approach followed in this paper, a process specification is assumed to be static, i.e., does not change during execution. This is, however, not and inherent requirement for TxProc/IDF. Considerable work addressing dynamic process evolution, in particular in the context of medical information systems, has been done in [19] and [16]. Since these concepts are orthogonal to our approach, they can be integrated without affecting the special functionality of TxProc/IDF.

While processes allow to rather loosely couple systems and provide for an application-centric approach, the work presented in [13] addresses a data-centric integration of distributed systems in order to access information in a transparent manner.

By exploiting the functionality of our prototype system WISE, we rely on the theory of transactional process management. Unlike approaches like spheres of joint compensation [14] which addresses only atomicity of processes, we treat concurrency control and recovery jointly (similar to ConTracts [23]). In addition, we make intensive use of the special process structure supported by transactional process management. In particular, the presence of non-compensatable activities and appropriate alternative executions which lead to the provable correctness of single processes are extremely important in environments where reliability is a major concern (such as medical information systems); this functionality, however, cannot be found in related approaches.

Since we study processes and logical application systems, we focus on the top two layers of the 3–layer model for medical information systems [25]. The physical architecture of the application systems is hidden behind communication server and TCAs.

Distributed workflow management systems often consist of a large numbers of applications and services. [11] presents a formal method for configuring a distributed system such that the processes' demands regarding performance and availability can be met while, at the same time, aiming at minimizing the total system costs.

7 Conclusion

In this paper we have discussed how two different paradigms, namely the transactional support for complex processes and data transfer and conversion can be brought together. Based on a rather simplified process taken from the practice of medical information systems, we have shown how complex the requirements with respect to control flow and data flow within this environment can be.

Communication servers help in transferring and transforming data between different systems, but do not provide any support for global control in distributed heterogeneous systems. To cope with the latter problem, the paper introduces the theory of transactional process management and motivates why traditional transaction models, as used in banking and industrial environments, are not sophisticated enough to satisfy the requirements of medical information systems. TP–Monitors impose too strong restrictions on long–running and dynamic processes. Open nested transactions solve this problem, but only provide for a limited form of failure handling that is restricted to simple and highly unrealistic cases where each step in the medical process would have to be compensated in the case of failure.

Finally, we have presented the prototype system TxProc/IDF we have implemented at ETH. TxProc/IDF combines the functionality of a commercial communication server with that of the transactional process management system WISE, which has been developed at ETH in a joint effort by the information and communication systems group and the database research group. Monitoring of processes, although being an integral requirement for the practical use of such a system, has not been discussed in this paper. Basic monitoring functionality is already integrated in the WISE system; this allows even to "zoom" into underlying applications in order to gather information on the state of the service invoked there. We have recently completed a prototype implementation [17] allowing the monitoring of SAP R/3 Business Processes that are executed as activities of the WISE system directly from the monitoring interface of WISE. In our future work, we aim at extending this functionality in order to support a broader range of applications that can be observed with this paradigm. In addition, we also plan to extend these monitoring facilities to data which is located in the external temporary persistent storage of TxProc/IDF. A further extension of the TxProc/IDF system will be the support for dynamic data routing. Each TCA should have the possibility to either sent data directly back to the WISE system or to apply the indirection via communication server and temporary persistent storage. The decision on which strategy is to be used then depends on the size of the data objects to be transferred.

References

[1] e*gate, Software Technologies Corperation (STC), http://www.stc.com.

[2] Healthcare.com, formely HIE, represented by Health–Comm GmbH, http://www.healthcare.com.

[3] IvyTeam , http://www.ivyteam.ch.

[4] G. Alonso, U. Fiedler, C. Hagen, A. Lazcano, H. Schuldt, and N. Weiler. WISE: Business to Business E-Commerce. In *Proceedings of the 9th International Workshop on Research Issues in Data Engineering. Information Technology for Virtual Enterprises (RIDE-VE'99)*, Sydney, Australia, March 1999.

[5] G. Alonso, R. Vingralek, D. Agrawal, Y. Breitbart, A. El Abbadi, H.-J. Schek, and G. Weikum. Unifying Concurrency Control and Recovery of Transactions. *Information Systems*, 19(1):101–115, 1994.

[6] Th. Bauer and P. Dadam. Efficient Distributed Workflow Management Based on Variable Server Assignments. In *Proceedings 12th Conference on Advanced Information Systems Engineering*, pages 94–109, Stockho lm, Sweden, S., 6 2000.

[7] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[8] P. Bernstein and E. Newcomer. *Principles of Transaction Processing*. Morgan Kaufmann Publishers, 1997.

[9] A. Elmagarmid, editor. *Database Transaction Models for Advanced Applications*. Morgan Kaufmann, 1992.

[10] H. Garcia-Molina. Using Semantic Knowledge for Transaction Processing in a Distributed Database. *ACM Transactions on Database Systems (TODS)*, 8(2):186–213, June 1983.

[11] M. Gillmann, J. Weißenfels, G. Weikum, and A. Kraiss. Performace and Availability Assessment for the Configuration of Distributed Workflow Management Systems. In *Proc. of the 6th Intl. Conference on Extending Database Technology (EDBT '00)*, pages 183–201, 2000.

[12] J. Gray, A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.

[13] J. Grimson, W. Grimson, and W. Hasselbring. The SI Challenge in Health Care. *Communications of the ACM (CACM)*, 43(6):48–55, 6 2000.

[14] F. Leymann. Supporting Business Transactions via Partial Backward Recovery in Workflow Management Systems. In *Proceedings of Datenbanksysteme in Büro, Technik und Wissenschaft (BTW'95)*, Informatik Aktuell, pages 51–70, Dresden, Germany, March 1995. Springer Verlag.

[15] S. Mehrotra, R. Rastogi, A. Silberschatz, and H. Korth. A Transaction Model for Multidatabase Systems. In *Proceedings of the 12th International Conference on Distributed Computing Systems (ICDCS'92)*, pages 56–63, Yokohama, Japan, June 1992.

[16] R. Müller and E. Rahm. Rule-Based Dynamic Modification of Workflows in a Medical Domain. In *Proceedings of Datenbanksysteme in Büro, Technik und Wissenschaft (BTW'99)*, Informatik Aktuell, pages 429–448, Freiburg, Germany, March 1999. Springer.

[17] A. Naef, Ch. Schuler, and H. Schuldt. Monitoring of Complex Services in Inter–Enterprise Processes with SAP R/3 Business Workflows. In *Proceedings of BTW 2001*, Oldenburg, Germany, March 2001.

[18] R. Orfali, D. Harkey, and J. Edwards. *The Essential Client/Server Survival Guide*. John Wiley & Sons, second edition, 1996.

[19] M. Reichert and P. Dadam. ADEPT$_{\text{flex}}$ — Supporting Dynamic Changes of Workflows Without Losing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, March 1998.

[20] H. Schuldt, G. Alonso, and H.-J. Schek. Concurrency Control and Recovery in Transactional Process Management. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS'99)*, Philadelphia, Pennsylvania, USA, May/June 1999.

[21] H. Schuldt, H.-J. Schek, and G. Alonso. Transactional Coordination Agents for Composite Systems. In *Proceedings of the International Database Engineering and Applications Symposium (IDEAS'99)*, Montreal, Canada, August 1999.

[22] R. Vingralek, H. Hasse-Ye, Y. Breitbart, and H.-J. Schek. Unifying concurrency control and recovery of transactions with semantically rich operations. *Theoretical Computer Science*, (190):363–396, 1998.

[23] H. Wächter and A. Reuter. *The ConTract Model*, chapter 7. In: [9]. 1992.

[24] G. Weikum and H. Schek. Concepts and Applications of Multilevel Transactions and Open Nested Transactions, chapter 13. In: [9]. Morgan Kaufmann Publishers, 1992.

[25] A. Winter. Integration of Application Systems into Hospital Information Systems – Strategy and Experiences, 1996. In German.

[26] A. Winter, H. Janßen, E. Glück, R. Haux, and J. Wiederspohn. Zur verteilten Datenverarbeitung bei heterogenen Subsystemen am Beispiel des Heidelberger Klinikuminformationssystems. In *GI Jahrestagung 1990*, pages 232–241, Stuttgart, Germany, October 1990.

[27] A. Zhang, M. Nodine, B. Bhargava, and O. Bukhres. Ensuring Relaxed Atomicity for Flexible Transactions in Multidatabase Systems. In *Proc. SIGMOD '94*, pages 67–78, 1994.