Abramowicz, W.; Orlowska, M.E. (Eds.): BIS 2000 - Proc. of the 4th Int'l Conference on Business Information Systems, Poznan, Poland, April 2000, Springer-Verlag, 2000, pp. 36-59

Clinical Workflows - The Killer Application for Process-oriented Information Systems? ♦

Peter Dadam, Manfred Reichert
University of Ulm
Dept. Databases and Information Systems
89069 Ulm, Germany
{dadam, reichert}@informatik.uni-ulm.de

Klaus Kuhn
University of Marburg
Dept. Medical Informatics
35037 Marburg, Germany
kuhn@ mailer.uni-marburg.de

Abstract

There is an increasing interest in changing information systems to support business processes in a more direct way. Workflow technology is a very interesting candidate to achieve this goal. Hence the important question arises, how far do we get using this technology. Is its functionality powerful enough to support a wide range of applications or is it only suitable for rather simple ones? And, if the latter is the case, are the missing functions of the "just to do" type or are more fundamental issues addressed? The paper uses the clinical domain to motivate and to elaborate the functionality needed to adequately support an advanced application environment. It shows that workflow technology is still lacking important features to serve this domain. The paper surveys the state of the art and it presents solutions for some issues based on the concepts elaborated in the ADEPT project.

1. Introduction

For a variety of reasons, companies are developing a growing interest in changing their information systems (IS) such that they behave "process-oriented". That means to offer the right tasks, at the right point in time, to the right persons along with the information needed to perform these tasks. Up to now, most IS are (more or less) purely functionally oriented: An application system, usually sitting upon a database system, is offering functions for querying customer data, for querying or entering order data, for writing invoices, etc. Making such systems to behave process-oriented is – if it is done by conventional programming methods (decision tables, state and transition tables, or if-then-else-statements) – a non-trivial task. As not all functions may be offered by one application system, it may even require to issue procedure calls across application modules, maybe even across different computers, thus making the implementation complex and error-prone. The most severe problem, however, is probably caused by program maintenance. While functionally oriented IS tend to remain stable for rather long periods of time, process-oriented IS have to be modified whenever the business process changes, and this may happen rather frequently. Once an IS has been made to behave strictly process-oriented, it must be adjustable to process changes and to evolving organizational structures very quickly and at reasonable costs.

When analyzing the alternatives to implement process-oriented IS in sufficient depth [9, 24], one comes to the conclusion that such systems can only be realized in a cost-effective and reliable fashion if the process (control) flow can be (1) described and implemented separately from the application functions and (2) the implementation of the application functions can be kept (nearly) as simple as in the case of purely functionally oriented IS. That is during implementation one should not have to bother about concurrency control and recovery (including logging) issues, about inter-process communication (e.g. remote procedure calls which do never return) and – ideally – also not about the issue that the order in which steps are performed in the process may be changed at a later point in time. That is it should be possible to implement application functions as isolated components which can expect that their input parameters are provided upon invocation by the run-time environment and which only have to worry about producing correct values for their output parameters. This is, when looking at the whole picture, not as trivial as it looks like at first glance! All the other issues should be handled by the build-time and run-time environment of the process-oriented IS.

If this can be achieved, then the implementation of process-oriented applications or whole process-oriented IS may one day consist of selecting an appropriate process or workflow (WF) template from a WF library, customizing the template according to the individual needs, picking-up appropriate application components from a component library, plugging them into the WF template, and running a system check whether the interfaces of the components harmonize such that they can properly work together as specified by the WF template [9]. This may

♦ This paper is a revised and extended version of [10].

also require an automatic or semi-automatic mapping process to achieve the ”interoperability” of these pre-existing (perhaps purchased) components.

(Process-oriented) WF technology, as offered by workflow management systems (WfMS) like FlowMark or WorkParty, is already providing a major step towards this direction. Advanced WfMS of this category allow to model the control flow and the data flow explicitly and independently from the implementation of the application components. They “deliver” the values for the input parameters to the component before its invocation and pick-up the output values after its completion. This is done in FlowMark, for example, by so-called input/output containers which allow, in principle, to write the application components in an isolated fashion as described above. Hence the interesting point is how far do we get with such kind of technology (database technology for data management + WF technology for WF management + application functions) for the implementation of process-oriented IS when being applied in real-world application environments? Does this technology cover a broad spectrum of application areas or is it narrow in the sense that only a certain type of applications can be supported adequately? And, if this is the case, which additional functionality is needed?

Based on many years of first-hand knowledge of and personal working experience in the clinical domain, based on the experiences made in dedicated projects [9, 17, 31], and having also insights and experiences in other domains like Business Administration and Engineering Processes, we are convinced that the clinical application domain is very valuable for the evaluation of technologies for real-world, large-scale process-oriented IS. We believe (and will explain this in the sequel) that the realization of process-oriented clinical IS is a great challenge – if not even the “killer application” for this type of technology. It combines, like in a nutshell, all the problems and challenges usually only found in different application areas. On the other hand, once the technology has been made powerful enough to adequately support this domain, it will be able to support a broad spectrum of different application areas. Because of this, clinical applications can serve as an ideal test bed for process-oriented IS.

In the next section we describe the characteristic properties of clinical working environments as far as they are relevant for this paper. It is a real life description and not an artificial scenario to justify a special research topic. In Section 3 we evaluate the description of this application scenario, and we discuss and derive technological requirements. Section 4 gives a survey on research and development contributions. In Section 5 we show that some of these aspects should be considered in conjunction with each other and we use the ADEPT concepts to illustrate how an integrated approach could look like. Section 6 concludes with a summary and an outlook.

2. Clinical Working Environments

In hospitals, the work of physicians and nurses is burdened by numerous organizational as well as medical tasks. Medical procedures must be planned and prepared, appointments be made, and results be obtained and evaluated. Usually, in the diagnostic and treatment process of a patient various, organizationally more or less separate units are involved. For a patient treated in a department of internal medicine or surgery, for example, tests and procedures at the laboratory and the radiology department become necessary. In addition, specimen or the patient himself have to be transported, physicians from other units may need to come and see the patient, and reports have to be written, sent, and evaluated. Thus, the cooperation between organizational units as well as the medical personnel is a vital task, with repetitive but nevertheless non-trivial character. Processes of different complexity and duration (up to several months) can be identified. One can find short processes, like order entry and result reporting for radiology, but also complex and long-running (even cyclic) treatment processes like chemotherapy for in- and out-patients.

Physicians have to decide which interventions are necessary or not – under the perspective of costs and invasiveness – or which are even dangerous because of possible side-effects or interactions. Many procedures need preparatory measures of various, sometimes considerable complexity. Before a surgery can take place, for example, a patient has to undergo numerous preliminary examinations, each of them requiring additional preparations. While some of them are known in advance, others may have to be scheduled dynamically, depending on the individual patient and his state of health. All tasks may have to be performed in certain orders, sometimes with a given minimal or maximal time distance between them. After an injection with contrast medium was given to a patient, for example, some other tests cannot be performed within a certain period of time. Usually, physicians have to coordinate the tasks related to their patients manually, taking into account all the dependencies existing between them. Changing a schedule is not trivial and requires time-consuming communication. For some procedures, physicians from various departments have to work together. So, coherent series of appointments have to be arranged, and for each step actual and adequate information has to be provided. Typically, each unit involved in the patient treatment process concentrates on the function it has to perform. Thus, the process is subdivided into partial, function- and organization-oriented views, and optimization usually stops at the border of the department. For all these reasons many problems result:

- Patients have to wait, because resources like physicians, rooms, or technical equipment are not available.
- Medical procedures may become impossible to perform, if information is missing, preparations have been omitted, or if a preceding procedure has been postponed, canceled or requires latency time. Subsequent appointments may then have to be changed as well, which results in intensive phone-calls and time losses.
- If any results are missing but urgently needed, tests or procedures may have to be performed repeatedly.

Because of this, from the patient as well as from the hospital perspective unpleasant and unwanted effects occur: Hospital stays can be longer than necessary and the costs or even the invasiveness of the patient treatment may increase. In critical situations, missing information may lead to late or even wrong decisions. Investigations have shown that medical personnel is aware of these problems and that computer systems helping to make appointments and providing the necessary information would be highly welcome by nurses and physicians. In an increasing way it is being understood, that the correlation between medicine, organization and information is high, and that today's organizational structures and IS offer only sub-optimal support. This is even more the case for hospital-wide and cross-hospital processes and for health care networks.

The roles of physicians and nurses complicate the problem. They are responsible for many patients and they have to provide an optimal treatment process for each of them. Medical tasks are critical to patient care and even minor errors may have disastrous consequences. The working situation is further burdened by frequent context switches. Physicians often work at various sites of a hospital in different roles. In many cases unforeseen events and emergency situations occur, patient status changes, or information necessary to react is missing (up to: "where is my patient?"). In addition, the physician is confronted with a massive load of data that have to be structured, intellectually processed, and put into relation to the problems of the individual patient. One can show that physicians tend to make mistakes (e.g., wrong decisions, omissive errors) under this "informational overload" [23].

From the perspective of a patient, a concentration on his treatment process is highly desirable. Medical personnel, very similarly, wishes to treat and help patients and not to spend their time on organizational tasks. From the perspective of health care providers, the huge potential of the improvement of clinical processes has been identified: length of stay, number of procedures, number of complications could be reduced. Hence there is a growing interest in process orientation and quality management. Medical and organizational processes are being analyzed, and the role of medical guidelines describing diagnostic and treatment steps for given diagnoses is emphasized.

When efforts are taken to improve and to automate the flow of clinical processes, it is extremely important not to restrict the physician or the nurse. Early attempts to change the function-oriented views of patient processes have been unsuccessful whenever rigidity came with them. Variations in the course of a disease or a treatment process are deeply inherent to medicine; the unforeseen event is to some degree a "normal" phenomenon. Medical personnel must be free to react and is trained to do so. In an emergency case, for example, physicians may collect information about a patient by phone and proceed with the process, without waiting for the (electronic) report to be written. A medical procedure may have to be aborted if the patient's health state gets worse or the provider finds out that a prerequisite is not met. Such deviations from the pre-planned process are frequent and form a key part of process flexibility. A computer-based system which is used to assist physicians and nurses in their daily work, therefore, must allow users to gain complete initiative whenever they need it. It must be easy to handle, self-explaining and – most important – its use in exceptional situations should be not more cumbersome and time-consuming than simply handling the exception by a telephone call to the right person; otherwise the system won't be accepted by the medical personnel.

The described working scenario makes it evident that clinical processes are of considerable complexity and of long duration. A hospital must provide an optimal treatment for hundreds up to thousands of patients within the same period of time. Already for a single treatment process, numerous organizational units – or even different hospitals and health care providers – may be involved in. The adequate support of hospital-wide and cross-hospital processes, therefore, can be considered as a great challenge for future process support systems.

3. Technological Challenges for Process-oriented (Clinical) Information Systems

The described scenario illustrates that an IS giving instant access to an electronic patient record would be very valuable; it would help to save time and thus help to reduce stress for the personnel. It makes also evident, however, that the online access to patient data alone will not dramatically improve the situation. At first glance, knowledge-based systems, "fed" with comprehensive medical knowledge could help a lot. We have also made experiments into this direction [17]. The identified problems are (as often) not at the technological side; they are knowledge-acquisition, knowledge-maintenance, and potential legal problems with respect to responsibilities. Our

conclusion was that a clinical IS should concentrate on the support of organizational procedures; i.e., it should assist the personnel in doing their work at the right point in time and, by providing the appropriate information, support them in taking the right decisions. But even if one concentrates on organizational procedures, the problem of "smooth assistance" still occurs. For clinical processes, it is nearly impossible to pre-model all possible task sequences and all exceptions in advance. If a physician, for example, comes to the conclusion that an additional measure which has not been anticipated in the process template becomes necessary in a given situation, the process-oriented IS must not prevent him to perform this measure. Otherwise, the IS would have to be "bypassed", which would cause (besides other problems) the problem of missing documentation.

Taking all these aspects together (and, in addition, some aspects not explicitly mentioned above) one can state the following list of requirements for a process-oriented clinical IS:

1. It must offer a WF meta model which is expressive enough to capture all relevant aspects of a process in an integrated and consistent way; e.g., control/data flow, temporal constraints, or organizational responsibilities
2. High reliability and consistency are key requirements for such a system. The WF model must enable formal verifications for the absence of any obscure system behavior at run-time (e.g., deadlocks, lost updates, invocation of task programs with wrong or missing input parameters, or temporal inconsistencies).
3. It must support ad-hoc deviations from the pre-modeled WF template at the WF instance level (i.e., to omit or postpone steps, to change the sequence of steps, or to insert new steps). Such dynamic changes must not violate the data consistency, the specified temporal constraints, or the robustness of the system. In addition, all changes must be properly integrated, especially with respect to authorization and documentation.
4. To deviate from the WF template must not be complicated for users. All the complexity associated with the re-mapping of the input/output parameters of the components affected by a change, the problem of missing input data due to the deletion of steps, or the problem of deadlocks must be hidden to a large degree from them. Instead, users must be able to define a dynamic change at a high semantic level, without requiring that they are familiar with a WF description formalism or a WF editing tool. In order to be able to realize application-specific user interfaces, the system must offer advanced programming interfaces to developers.
5. The system must support temporal aspects (cf. Sect. 2). – Besides deadlines, it should know about the externally fixed dates for steps and about temporal dependencies (e.g., minimal / maximal time distances) between them. It should monitor the WF execution with respect to these constraints and inform the user (in a reasonable way, no "window terrorism") about potential problems during run-time (e.g., due to delays).
6. For the efficient support of hospital-wide and cross-hospital processes, scalability of the process support system is a must. The system must run with acceptable performance, even if the number of users and concurrently active WF instances becomes very large.

Although already rather lengthy, this list is still incomplete. Further important aspects concern the evolution of WF schemes (incl. the adaptation of in-progress instances), the handling of inter-workflow dependencies, the treatment of "media-breaks", or the problem of mobility with respect to data entry and retrieval. Our experiences led to the insight that any serious attempt to develop a process support system should not only concentrate on isolated aspects, but must try to look (as far as possible) at the overall picture. As shown in Sect. 2, for example, the WF meta model should support cyclic tasks. This does not look very complicated at first glance. In conjunction with dynamic changes, however, it is not trivial at all, as we will show in Sect. 5.3. The same applies to temporal constraints. If both, dynamic changes and temporal constraints, are supported, then the addition of a new step or the change of pre-modeled task sequence, may lead to temporal inconsistencies (cf. Sect. 5.4). Finally, the support of dynamic changes should not negatively influence the performance of the system. Concepts which have been developed to achieve scalability must also work in conjunction with dynamic changes (see Sect. 5.5).

4. Contributions from Research and Development

The separation of the flow structure from the implementation of the application components has been already realized (at least partially) in systems like FlowMark and WorkParty. Today's high-end WfMS, however, have been primarily designed for the support of well-structured processes showing little variations in their possible task sequences. They implicitly assume that all aspects of a process and all tasks are known in advance, and they usually enforce a strict execution of the pre-modeled WF template. Although on-the-fly modifications of organizational entities or substitutions of task components during run-time are supported, these WfMS are rather weak with respect to dynamic WF changes. There are some (document-centered) systems that allow the user to deviate from the pre-modeled WF template at run-time, but at the risk of inconsistencies and errors.

To cope with the combinatorial problem of pre-modeling all possible execution paths at build-time, different mechanisms have been suggested. MOBILE [18] provides a *descriptive WF model*. It allows the designer to omit those aspects of a WF model (e.g., the order of tasks), which may be defined by end users at run-time. Although this approach allows to combine tasks in a very flexible way, it is only applicable as long as the task programs are encapsulated and autonomous, so that they may be executed in an arbitrary order. HOON [16] follows another approach by supporting the *late binding* of task resources (e.g., task programs and sub-workflows). It does not support dynamic changes, however, since structural adaptations of a WF model are not possible after it has been bound to a task instance. A more advanced approach is offered by MOVE [15], which supports the concept of *late modeling*. The designer may specify task nodes for which a sub-workflow may be dynamically defined or modified. Late modeling, however, requires that users are familiar with the used WF language.

There are several approaches that support *ad-hoc deviations* from the pre-modeled WF template at the WF instance level. Many of them are based on the *object migration model*. In this model, a WF instance (together with its definition) is regarded as an object ("circulation folder") which is sent from actor to actor according to the modeled flow [20]. Only the user who is currently in charge of the folder may change the flow structure, e.g., by adding an intermediate task. This approach may be sufficient for the coordination of document-centered WFs, but not for the support of clinical WFs. A potential weakness is the simplicity of the used WF model. Parallel branchings and loop backs are not directly supported. The offered change operations do consider the control flow, but ignore other aspects of the WF model, like the data flow or temporal constraints. This leaves significant complexity to application programmers, who themselves must ensure that the correctness of the WF is preserved when a change is performed. There are several approaches which combine graph-based WF models with *rule-based extensions* for exception handling [19, 28]. In HematoWork [28], for example, global rules are used to enable the WfMS to automatically restructure the control flow of a WF instance at the occurrence of logical exceptions. The necessary adaptation of the data flow is not considered. In MOKASSIN [19] rules serve as the basis for extending the WF meta model by user-defined constructs as well as for configuring the WF behavior (e.g., with respect to dynamic changes). The most severe problem, namely rule maintenance and consistency, is not discussed.

There are only few approaches which address correctness issues in conjunction with dynamic changes [8, 14, 22, 25, 32]. Most of them concern the handling of in-progress WF instances when their schema is modified (*WF schema evolution*). Except TRAM [22], the proposals mainly deal with dynamic changes of the control flow, while adaptations of other WF aspects are not considered. In [14], a change corresponds to the replacement of a subnet of the WF graph by a new one. It is said to be correct if afterwards the WF instances can be either executed according to the old schema or to the new one. WIDE [8] and TRAM [22] provide a set of change primitives and evolution policies. Formal criteria are introduced in order to determine which WF instances can be transparently migrated to the new schema. Similar approaches are discussed in [25, 32]. – There are several other proposals for adaptive WFs, which come from different fields of computer science. Approaches which have been followed in the context of transactional WF models [24, 26], case-based reasoning [27], Petri nets [1, 3, 6], graph grammars [7], and configuration management [2, 6] are worth mentioning.

The WF meta models used by the discussed approaches capture only one or a few aspects of real-world processes, which may lead to problems when the WF structure is changed. The deletion or the addition of a task, for example, may cause data as well as temporal inconsistencies if no further precautions are taken. Due to the complexity of business processes, more comprehensive modeling techniques are needed, allowing to capture the richness and complexity that accompany their computer-based support. The challenge is to keep such techniques usable in practice and to keep the costs for model checking low. Although there exists a variety of formal models [14], adequate mechanisms for modifying structural components of a WF at run-time are missing. Finally, most approaches require that users are familiar with a WF modeling language or with a graphical modeling tool, respectively. Taking the scenario described in Sect. 2, however, one cannot expect that physicians and nurses under stress are able to change a WF instance graph by the use of a graph editor or a Petri net tool.

5. The ADEPT Approach to Process-oriented Information Systems

As already indicated, the main problem is that most of the issues addressed in Section 3 cannot be treated reasonably well when considered in an isolated fashion only. In the ADEPT project we are looking at different facets of process-oriented IS at the same time: user interfaces, exception handling, flexibility, dynamic changes, WF schema evolution, temporal aspects, inter-workflow dependencies, and scalability. Due to lack of space we cannot elaborate all interdependencies here. Instead, we focus on important issues arising in the context of dynamic WF changes. We use the ADEPT concepts to illustrate the problems and to outline how an integrated solution could look like. We only describe here those parts of the ADEPT methodology which are necessary for this discussion, however. More comprehensive treatments can be found in [4, 5, 9, 29, 30, 31].

5.1 The ADEPT Base Model and Usability Aspects

One of the challenges for process-oriented clinical IS is to find a reasonable compromise between the expressive power of the modeling language on the one side and usability by the WF designer as well as by the end-user on the other side. "Adequately" in this context means that the resulting model should represent the real-world process as naturally as possible. As shown in Sect. 2, for example, we have found the support of cyclic tasks to be important. Thus the modeling language should allow to express loops explicitly. "Usability" on the one hand means that a representation of the running WF instance is still understandable to the end-user, or at least to those users which may perform dynamic changes. That is, a user must be able to understand the consequences of a change he is going to perform and should also be able to understand why the system is refusing to perform a certain change request; i.e., there should be no "magic" behind. Usability on the other hand means, that the user must be sure that a dynamic change does not violate the consistency and robustness of the system. Such checks cannot be performed by users, especially not during the exceptional situation which has caused the dynamic change, but have to be performed by the system. This means that the WF model must allow to detect all inconsistencies and to perform the checks efficiently (think of an emergency case which causes a deviation from the standard process; one cannot wait for minutes for the system's decision in such a situation). For these reasons we have dismissed, for example, the idea of using Petri nets for WF modeling. Instead, we have adopted concepts from block-structured process description languages [13] and have enriched them by introducing further control structures (cf. Fig. 1) and correctness criteria regarding the data flow and the dynamic behavior of a process [30].

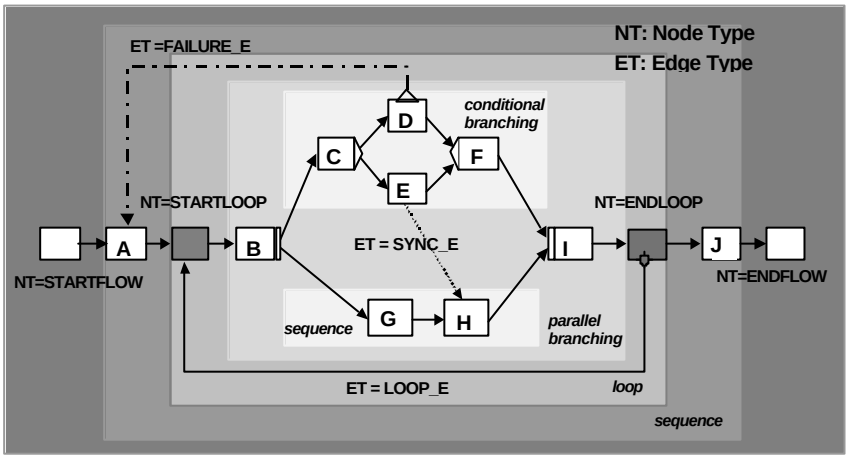

Fig. 1: *Modeling Processes in ADEPT – the Base Model*

Fig. 1 and Fig. 2 illustrate the philosophy how processes are modeled in ADEPT. This representation is not intended for the end-user. However, even more "end-user-friendly" interfaces will somehow reflect the basic concepts described here to avoid too large discrepancies between the mental model of the user and the model used in reality by the system. In the ADEPT WF meta model different types of nodes are used to discriminate between different kinds of branchings and loops. One class of edge types is used to describe the control flow and another one, together with so-called "data elements", to describe the data flow of the process (the data elements store the data versioned to allow partial rollback). In the ADEPT WF model branchings as well as loops are always modeled in a block-oriented fashion. A branching block as well as a loop block always has exactly one entry and one exit node. Blocks may be nested but are not allowed to overlap. As this limits the expressive power of the WF model so-called "synchronization edges" (see below) can be used in addition which allows to describe more complex structures, if necessary. We have selected this block structure because it is rather quickly understood by users, it allows the use of syntax-directed WF model editors, and it makes it possible to implement efficient algorithms for consistency checking. The "ingredients" described so far are the same for WF templates (as stored in the WF repository) and for WF instances in execution. At the instance level, special labels (ACTIVATED, RUNNING, ...; see Fig. 2) are used to exactly describe the status of nodes and edges during the execution of the WF instance. In addition, well-defined rules exist for changing these markings during WF execution.

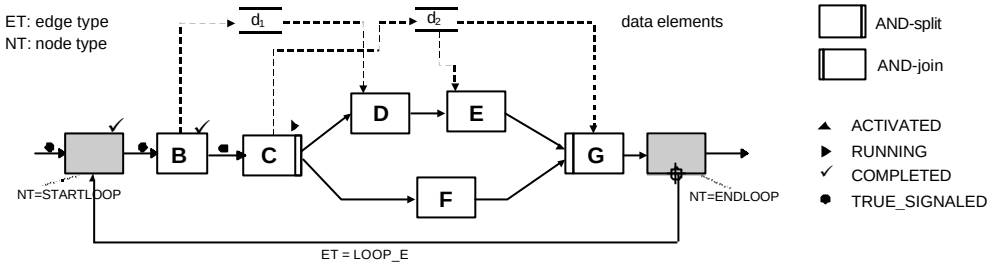

Fig. 2: *Example of a WF instance graph (partial view)*

5.2 Support of Dynamic Changes

As described above, the support of dynamic changes makes only sense, in general, if there is no risk of subsequent program crashes or data losses due to lost update problems. One approach to solve the problem is to only support trivial modifications like allowing to insert a new step into a sequential chain of steps or to enforce that all previous steps are completed before the newly inserted step can be activated. Such measures make consistency checking much more easier, but also restrict the practical usability of this feature significantly. In ADEPT we, therefore, decided to support all types of dynamic changes: insertions (as sequential or as parallel steps), deletions, and shifts (i.e., changing the sequence of steps). Such changes must lead again to a proper block structure, in which additional synchronization edges may be used, if necessary (see below). The user is not really burdened with this restructuring of the WF graph. He or she expresses the change request in a rather declarative way (e.g., to insert a new step between two node sets) and at a high semantic level. For this high-level, powerful modification operations are offered, which hide the complexity of a dynamic change to a large degree from the user; internally, graph transformation rules and graph reduction rules are used to perform the necessary transformation (see [30] for details). As response to a change request ADEPT, in essence, first checks all data dependencies and sequence constraints to detect whether the problem of missing input values, lost updates, or cyclic waits (deadlocks) may occur in the modified WF graph. In case of missing input values (which may occur due to the insertion, the deletion, or the shift of steps) ADEPT will offer the user to generate a form and prompt for these values (either immediately or when needed). Only if no consistency problem may occur – or if it is explicitly accepted by the user – ADEPT will accept the change request and perform the necessary graph transformations.

To illustrate the problems, let us assume that, at a certain point in time, the WF instance graph looks like as the one depicted in Fig. 2 (note that this representation is not intended for the end-user, who may define dynamic changes at a higher semantic level): The steps represented by the nodes Startloop and B are completed and step C is currently in execution. Let us further assume that an exceptional situation occurs which makes it necessary to immediately perform step D, but to maintain the order of all the other steps (i.e., step E after step D, step G after E and F; and transitively also after D). At first, the data dependencies for step D are checked. Step D is executable, in principle, because it receives its input data from step B which already has been completed and it does not produce any process relevant output data and therefore no problem of lost updates may occur. Thus the restructuring of the WF graph can be started. In order to make it possible that step D can be immediately started, step D must no longer be a successor to step C (because it would have to wait for the completion of C). Instead, D has to be placed in a parallel branch to step E. This means that the control edge from C to D has to be removed and replaced by a control edge from B to D.

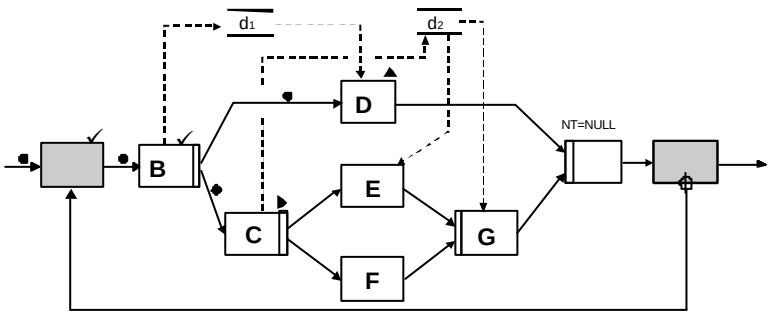

Fig. 3: *Dynamic Change - Intermediate Process Graph*

Fig. 3 illustrates how the process graph is looking like at this point in time. Step D has become a parallel step with respect to step C, its predecessor with respect to the control flow is now step B, and the control edge from B to D is marked with TRUE_SIGNALED which means that step D can be executed (because its start conditions are satisfied). This transformation alone would not be correct, however, because not all of the previously existing

constraints are obeyed any longer. It would be possible, for example, that step E is being started (once step C has been completed) in parallel or even before step D. To enforce the correct execution sequence, a "synchronization edge" (SYNC_E) from D to step E is introduced into the graph which enforces that step E cannot be started until step D has been completed. ADEPT uses different types of synchronization edges to express different kinds of "wait-for" situations (see [30]). The final process graph resulting from these modifications is depicted in Fig. 4.

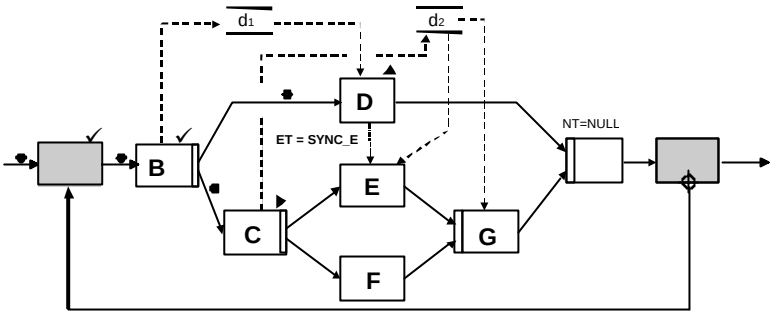

Fig. 4: *Dynamic Change - Final Process Graph*

5.3 Loops and Dynamic Changes

As motivated, we consider the provision of explicit loop constructs as being very important. It improves the readability of the WF model and allows to distinguish between unintentionally modeled cycles ("bad cycles") from intentionally modeled loops ("good cycles") during consistency checks. If both, loops and dynamic changes are supported in one system, then one has to decide on the duration of dynamic changes affecting steps which belong to a loop: Shall the modification be valid only for the current iteration of the loop (as it might be the case for the ad hoc deviation presented above) or shall it also be valid for all subsequent iterations? To "hard-wire" one of these alternatives is certainly no good idea. Instead, the system should offer to perform the modification temporarily (i.e., only valid for the current iteration) or to perform it permanently for this WF instance. To avoid potential misunderstandings with modifications of the WF template, we will refer to these modifications as loop-permanent and loop-temporary modifications in the sequel.

To support both, loop-temporary as well as loop-permanent modifications is not as straightforward as it looks at first glance, especially if they occur in combination. Additional rules are necessary to describe when a modification can become loop-permanent and when it can only become loop-temporary. The most important rule is that a loop-permanent modification must not depend on a previously performed loop-temporary modification. Otherwise it would either be lost in the next iteration (where the loop-temporary modification is no longer present) or it may cause inconsistencies or even program crashes. In addition, once a loop-permanent modification has been accepted, no additional user-interaction should become necessary during the subsequent loops to resolve consistency problems. How ADEPT is dealing with this situation is sketched in the following.

For each WF instance ADEPT potentially manages two process graphs. A process graph $P_{current}$, which reflects the currently valid flow structure (i.e., the set of nodes, edges, and data elements) as well as the status information of the process, and a process graph P_{perm} which describes the nodes, edges, and data elements that are loop-permanently valid. Note, that P_{perm} may contain more or less nodes and edges than $P_{current}$ depending on the kind of loop-temporary modifications which have been applied (in addition, all changes to this WF instance are kept in a "change history log" which records them with timestamps, duration of validity, starting conditions, etc.). These two graphs are serving as the basis for performing loop-temporary and loop-permanent modifications. Loop-permanent modification requests are first checked against P_{perm}. If no unresolvable conflicts are detected, then the modification request is checked against $P_{current}$ as well, because loop-temporary changes may prevent to accept the permanent change at this point in time. In such cases ADEPT will allow to "register" the change request and the change will become effective at the next possible point in time. – We illustrate the usage of the two graphs by an example: Assume our WF as described in Fig. 4 has completed steps C, D, E, and G. In addition, a loop-permanent deletion has been applied to step F and two new steps X and Y have been loop-temporarily inserted succeeding the steps D and G, but still within the loop. Furthermore, a step Z has been (permanently) inserted after the loop-end-node. The resulting $P_{current}$ graph could then look like at a certain instance in time as depicted in Fig. 5. The deleted node is now represented by a "gravestone" (NULL –node, i.e., a dummy activity without associated action) and the three newly inserted nodes, together with their data elements as well as control flow and data flow edges, are appearing. The corresponding P_{perm} graph would then look like the one depicted in Fig. 6. Steps X and Y are not present in this graph, but step Z is.

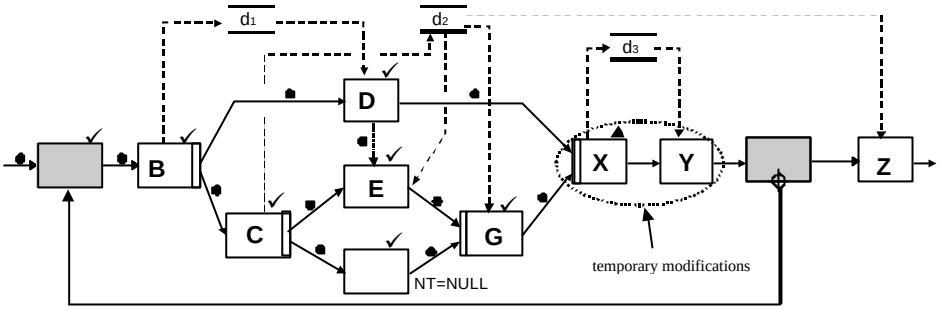

Fig. 5: $P_{current}$

The proper treatment of temporary changes is somewhat more complicated as may be derived from this simple example. Loops, for example, may be nested. Thus several levels of loop-temporary changes may occur. Think, for example, that the loop depicted in the figures above is framed by an outer loop and that step Z is not permanently inserted but is a loop-temporary modification which is performed while the system is in the "inner loop". Assume further that there is a data dependency between step X and step Z. What shall happen to step Z when the current iteration of the inner loop is completed and step X disappears again? One possibility is to treat Z like a newly inserted step and to prompt the user for the missing input values. Another possibility is to inform the user and to apply a "cascading delete" semantic whenever temporarily inserted steps are depending on other temporarily inserted steps which have been removed. This latter strategy is applied in ADEPT, for example.

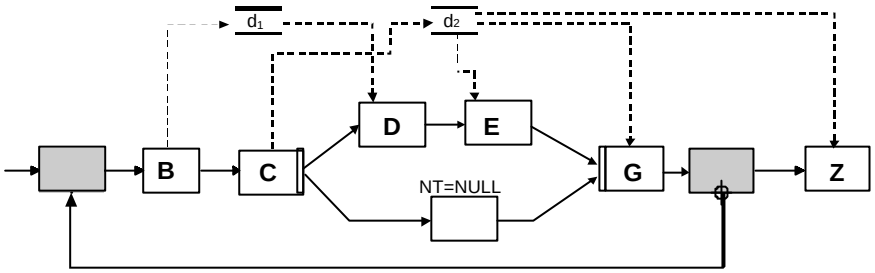

Fig. 6: P_{perm}

5.4 Temporal Constraints and Dynamic Changes

As motivated in Sections 2 and 3, the management of temporal constraints is an important feature of a process support system. Although some functionality for considering temporal aspects has been incorporated into existing WfMS, usually, it has been limited and has failed to capture the modeling requirements of real-world processes. We, therefore, enhanced the capabilities of the ADEPT base model (cf. Sect. 5.1) by providing advanced concepts for the definition and monitoring of temporal constraints; in the following we call this extension $ADEPT_{time}$. In addition to the modeling concepts described in Sect. 5.1 (see also [30]), $ADEPT_{time}$ allows the WF designer to express different kinds of temporal constraints with respect to a given WF template. For each WF activity, a minimal and a maximal duration may be specified. In addition, time dependencies between the activities of a WF graph can be defined (e.g.,"activity X must be completed 2 days before activity Y starts"). For their modeling $ADEPT_{time}$ introduces an additional edge type to the WF meta model: A *time edge* (cf. Fig. 7) connects two activities X and Y and defines a minimal or maximal time distance between them. $ADEPT_{time}$ supports four different time relationships: completion/start, start/start, completion/completion, and start/completion.

For a given WF template, $ADEPT_{time}$ already checks at build-time whether its temporal constraints are satisfiable; i.e., whether there exists at least one valid time schedule (i.e., an assignment of absolute starting and finishing times to the WF activities, such that all constraints are satisfied). At run-time, each node X of an instance graph is associated with additional temporal attributes: EST_X , LST_X, EFT_X, and LFT_X, representing its earliest and latest starting and finishing time. These attributes describe time intervals for the start and for the completion of the task X with the following semantics: If for an arbitrary node Z of the WF instance graph, a starting (finishing) time $ST_Z \in [EST_Z, LST_Z]$ ($FT_Z \in [EFT_Z, LFT_Z]$) is selected, then for all other nodes $Y \neq Z$ there also exists a starting time $ST_Y \in [EST_Y, LST_Y]$ and a finishing time $FT_Z \in [EFT_Z, LFT_Z]$ such that a valid time schedule results.

The time intervals of the WF activities are dynamically changed during WF execution: As soon as an activity is started (completed), its starting (finishing) time is fixed; i.e., we obtain time intervals of length 0. For activities which have not yet been started or completed, the earliest/latest starting and finishing times are automatically derived, so that the semantics described above is preserved. In addition, for pre-selected activities, the starting and

finishing time may be externally fixed by a WF client. The latter facility is useful for dealing with tasks (e.g., a medical examination) which are scheduled within an external (calendar) system. In any case, the fixed date should be consistent with the calculated starting and finishing times in order to meet the defined constraints. The calculation of the time intervals is done each time an activity is started or completed. In addition, it becomes necessary when a user assigns an external date to a task. For the checking of consistency and the calculation of the time intervals we use the Floyd-Warshall algorithm, similar to the approach described in [12]. The calculated time attributes are used for worklist presentation (e.g., to priorize tasks, to filter tasks, or to generate warnings) and for the proper selection of externally fixed dates. ADEPT monitors the WF execution with respect to the calculated times and it informs users if deadlines are going to be missed.

A simple example is depicted in Fig. 7. Activity B is in the state COMPLETED and activity C in the state RUNNING. Therefore, the starting and finishing time of B and the starting time of C have been already fixed. In addition, the starting time for activity G was fixed through an external appointment. All other time intervals were automatically calculated, considering the given temporal constraints (i.e., the minimal and maximal durations of the activities C, D, E, F, and G and the defined time distances between C and G and between F and G).

Already for the static case, time management is a non-trivial task. The ADEPT approach, however, supports both, dynamic WF changes and temporal constraints. Obviously, the consideration of temporal issues complicates the handling of dynamic changes, since a change must not only preserve the structural correctness and the data consistency of the WF instance graph, but also its temporal consistency. Think of, for example, the dynamic addition of a new step. In such a case, the time intervals of the WF activities must be re-calculated in order adapt their scheduling and to check whether any temporal inconsistency may occur. Assume, for example, that for the WF instance graph from Fig. 7, a user wants to insert a new step X between C and F. Depending on the duration of X, this change may cause a temporal inconsistency or not. If we have D_{min}^{X} = 1 h, for example, all time constraints are further met after the insertion of X. If, however, D_{min}^{X} is greater than 1 h, then the insertion of X will lead to a temporal inconsistency. In such a case, the ADEPT system informs the user about the conflict and offers him different strategies for resolving it, e.g., by aborting the change, by adapting the externally fixed starting interval of step G, by reducing the minimal duration of X, or by releasing other temporal constraints.

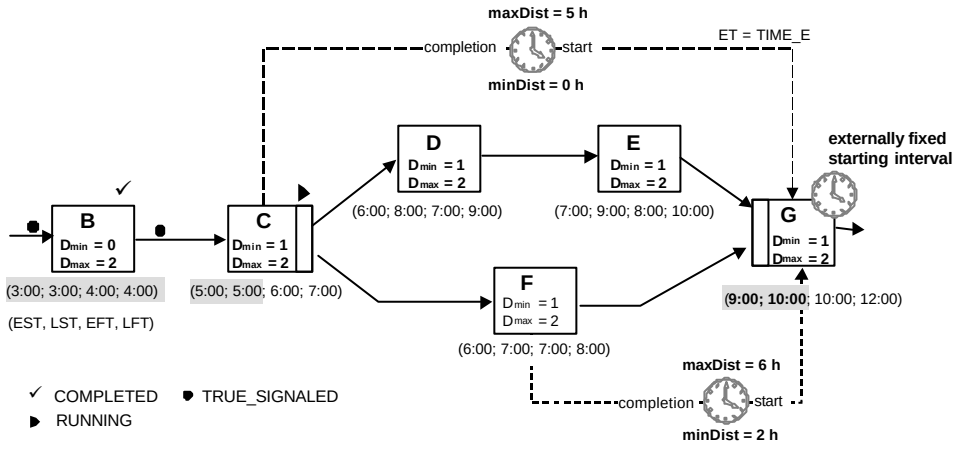

Fig. 7: *Management of Temporal Constraints in ADEPT$_{time}$ (The time edge between C and G (F and G) defines a minimal/ maximal time distance between the completion of C (F) and the start of G. The normal control edges are treated as (completion / start) time edges with minDist = 0 and maxDist = ¥)*

5.5 Distributed Workflow Execution and Dynamic Changes

Hospital-wide and cross-hospital WFs are characterized by a large number of users and concurrently active WF instances; i.e., a WF server may have to cope with a very high load. Already the processing of a single activity may require the transfer of multiple messages between a server and its clients, e.g., to transmit parameter data, to update worklists, or to invoke task programs. This may overload both, servers and subnets, if the number of concurrent WF instances increases. In addition, the units participating in a cross-organizational WF are often connected by slow wide area networks. Hence the load of the communication system is extremely critical for system performance. To keep communication local within one network segment, in many cases, it is advantageous to dynamically migrate the control of in-progress WF instances to a server in another network segment.

a) Distributed WF Control in ADEPT$_{distribution}$

For the efficient control of enterprise-wide and cross-enterprise WFs we developed the ADEPT$_{distribution}$ model [4, 5]. It supports the WF designer in partitioning a WF schema and in distributing the control of the partitions across several WF servers. The distribution is done in a way that prevents single system components (servers, network segments, and gateways) from becoming overloaded at run-time. At build-time, a WF schema is divided into several *partitions.* Each partition is assigned to a WF server, which then controls the corresponding activities at run-time. At each server a copy of the WF schema is stored. If a WF instance reaches a transition between two partitions, its control *migrates* to the WF server of the target partition. Before this server may proceed with the execution of the WF instance, WF data has to be transmitted as well. Following this approach, activities from parallel execution branches may be controlled by different WF servers at the same time. In order to keep communication costs low, ADEPT$_{distribution}$ does not require that these WF servers strictly synchronize with each other. Fig. 8 shows an example of a WF, which is controlled by multiple WF servers. In its current state, two servers – S_2 and S_3 – participate in the processing of this instance. Note that S_2 does not know the execution state of the lower branch, i.e., it does not know whether this branch is still controlled by S_1, the control has already been migrated to S_3 (as shown in the example), or has been given back to S_1 (in order to control the partition P_4).

ADEPT$_{distribution}$ partitions a WF schema in a way that minimizes the communication load of the system at run-time. The designer is supported by algorithms, which allow him to automatically calculate optimal server assignments for the activities of a given WF schema; a partition then consists of a subgraph of which the activities are assigned to the same WF server. To determine optimal server assignments, we use a formal cost model that allows us to evaluate the quality of a selected distribution. This model considers costs for the transfer of parameter data, for the update of worklists, and for the migration of WF instances. In most cases, WF activities are controlled by a WF server, which is located nearby their potential actors. Since migrations do also generate communication costs, however, they are only used if they improve the overall communication behavior of the system. In addition, we support *variable server assignments*, which enable the WfMS to select the server of a particular activity dynamically, depending on the *actor assignments* of preceding activities [(e.g., *Server(A_2)*:= *Domain(Actor(A_1))*; see [5] for details]. This does further improve the communication behavior of the system. More comprehensive treatments of the ADEPT$_{distribution}$ approach can be found in [4, 5].

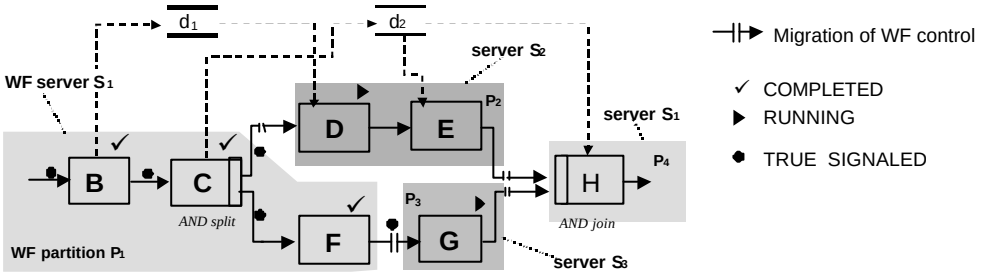

Fig. 8: *Distributed execution of workflows in ADEPT$_{distribution}$*

b) Dynamic Workflow Changes

So far, we have assumed that the structure of a WF instance graph is not changed during run-time. As shown, this is not realistic for real-world processes. The challenging question therefore is, how dynamic WF changes can be supported in a model like ADEPT$_{distribution}$, if we want to ensure the correctness and consistency of WF instances as in the central case and if we do not want to loose the advantages offered by the distributed control of WFs. In this context, it is important to minimize the synchronization effort which becomes necessary when a dynamic change is performed. A naive solution, therefore, would be to synchronize all servers that have already been involved in the processing of the WF instance or that may become active in the future. Generally, such a strict synchronization is not required and – in the case of variable server assignments – it is also not feasible.

We will illustrate some of the issues that arise in this context by an example. Taking the change operations provided by ADEPT [30] and the WF instance graph from Fig. 8, it is possible to dynamically insert a new activity X into this graph, whose execution may not start before step G is completed and which must terminate before activity E can be activated. Internally, this change is realized by applying a well-defined set of graph transformation and reduction rules. After its insertion, X constitutes a new branch of the parallel branching defined by the nodes C and H. The desired execution order (X after G, X before E) is enforced by inserting the two sync edges G$\rightarrow$ X and X $\rightarrow$ E (cf. Fig. 9). Assume that this change is initiated by a client connected to the WF server S_3. Obviously, the desired modifications are only allowed, as long as E has not been started. In order to check this, first of all, the WF server S_3 must retrieve the current state of activity E from the WF server S_2 (Note that S_3 itself does not know

the state of the upper branch). Conversely, the change must not only be applied to the WF instance graph stored at S_3, but it must also be considered for the copy of this graph stored at S_2. The latter becomes necessary in order to ensure that S_2 delays the execution of E until the newly inserted step X will be completed.

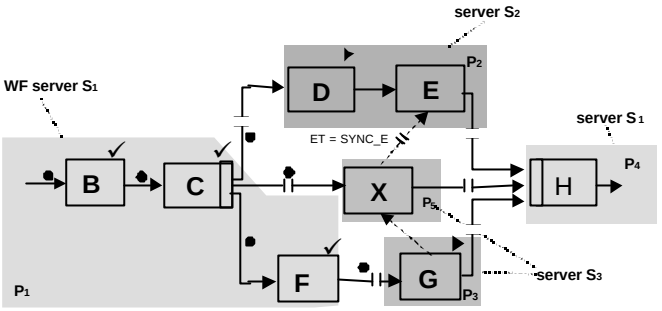

Fig. 9: *The WF instance graph from* **Fig. 8** *after a dynamic change*

WF servers that may become active in the following (like S_1 in our example) must also be informed about the change in order to be able to correctly proceed with the flow and to provide a proper basis for subsequent changes. This notification, however, must not be done immediately, i.e., at the time the change is introduced. Instead it is sufficient to transmit the relevant information, when the control of the WF instance migrates to this server. This causes additional communication costs for "normal" migrations, which should be kept as minimal as possible. In our approach, for example, we do not transfer the complete description of the modified WF instance graph to the target server. Instead, we use execution and change histories to reduce the communication amount. When migrating the control from S_2 (or S_3) to S_1 in our example (cf. Fig. 9), in addition to WF control data and WF relevant data, the relevant entries from the change history are transmitted as well. The WF server S_1 then applies the structural modifications to its local copy of the WF instance graph, before the execution may proceed. As far as possible, ADEPT tries to avoid the transfer of redundant change information.

Similarly to the execution of parallel branches, it is desirable to perform dynamic changes of single branches without costly communication with other WF servers. As shown in the example, for changes that affect multiple partitions of a WF instance graph this will not always be possible. Instead, it must be ascertained for how long and in which mode the WF instance has to be locked at the respective WF servers. As far as possible, long-term locks are avoided, so that the WF execution is not blocked unnecessarily long. There are numerous examples for dynamic WF changes, for which such a strict synchronization does not become necessary. Taking the WF instance graph from Fig. 8, for example, the WF server S_2 might insert a new activity between D and E without synchronizing this change with S_3. This is possible, since S_2 does not require information about the state of the lower branch in order to apply the change. Conversely, S_3 must not be informed about the modification of the upper branch in order to proceed with the control. Such local modifications do occur often in practice. Therefore, it does make sense to differentiate between different classes of changes with respect to a WF instance graph and to offer optimized procedures for their application. – Several other important issues are discussed in [29].

5.6 The ADEPT$_{workflow}$ Prototype

Since 1997, we have developed the advanced WfMS prototype ADEPT$_{workflow}$ which is based on the concepts described above. Its components are implemented in Java; for communication Java-RMI is used. The current version already supports dynamic changes, the management of temporal constraints, and the distributed control of WF instances. For demonstration purposes we use a WF client, which does not only show worklists to users, but which also visualizes the WF instance graphs, e.g., after a dynamic modification or a migration took place. In addition, the prototype comprises several build-time components. WF templates can be defined with the syntax-driven *ADEPT$_{workflow}$ process modeler*, which supports a variety of correctness and consistency checks. Complex organizational entities and relationships can be managed by the use of the *ADEPT$_{workflow}$ organization modeler*.

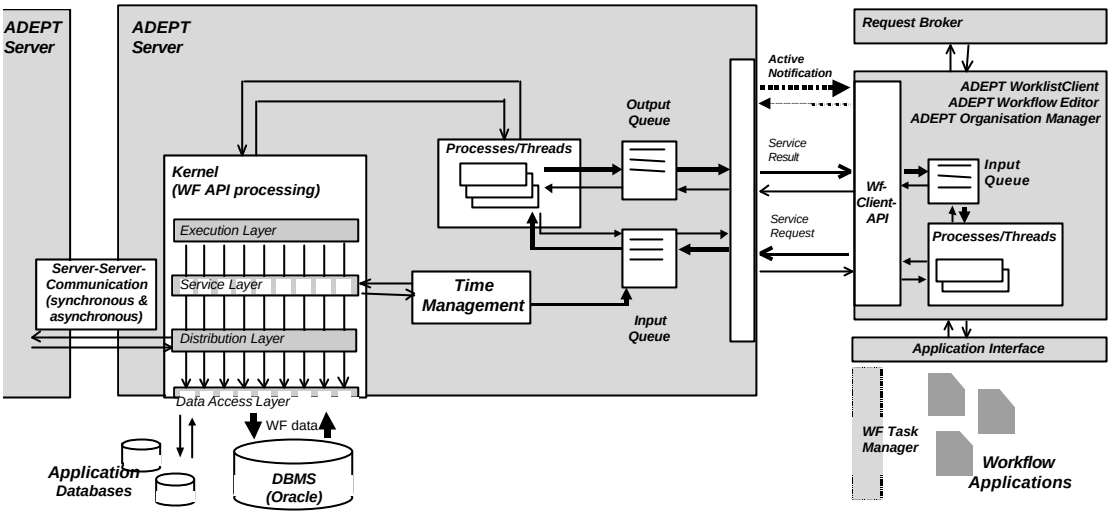

Fig. 10: *System Architecture of ADEPT$_{workflow}$*

The multi-server-architecture of ADEPT$_{workflow}$ is shown in Fig. 10. Different clients may be connected to a WF server; e.g., worklist handlers, monitoring components, or tools for the definition of WF templates and organization models. For the implementation of such WF clients, ADEPT offers a rich API with a functionality that goes far beyond to that of the WfMC standard interfaces [33]. Extensions became necessary to provide the advanced ADEPT features to application programmers. Furthermore, we have extended the one-directional client-server-communication in order enable WF servers to play an active role by initiating requests to clients (e.g., to get approvals from WF participants when performing a run-time deviation or to immediately notify them afterwards). The kernel of the server is realized as a multi-layer architecture. The top level, the *Execution Layer*, processes client API calls. Each call is decomposed into a set of service requests from the underlying *Service Layer*. This layer comprises services for the management of WF templates and instances, worklists, organizational entities, or temporal constraints. As an example consider the completion of an activity by a client. This leads to an update of temporal attributes as well as of the state of the WF instance, a role resolution for subsequent steps, and an update of worklists. Each service call itself is decomposed into several basic operations from the *Data Access Layer* (e.g., to read, create, or modify WF objects). If a migration of the WF control becomes necessary, in addition, the *Distribution Layer* provides the required data and performs the migration.

6. Summary and Outlook

The main issue of this paper was to discuss the challenges for process-oriented IS for real-world environments and to elaborate that these challenges respectively the technological answers cannot be treated in an isolated fashion but have to be understood in conjunction. We have used the clinical domain because we believe that it is indeed very challenging with respect to its functional needs on the one side, but it is not so “exotic” on the other side that the features needed there would not be very useful for other advanced application areas as well. We have described the clinical application scenario rather detailed to make clear under which working conditions WF technology must prove its usefulness and applicability, and to make also evident, that the discussed requirements are not just artificial ones. Based on these analyses the answer to the question posed in the title should be clear: it is “Yes, when looking at the whole picture, at present, clinical WFs are the 'killer application' for the process-oriented IS technology which is currently available”.

The discussion on supporting dynamic changes in conjunction with loops, temporal constraints, and distributed WF execution has shown that many non-trivial interdependencies among the different features exist, which must be carefully analyzed and understood. Similar like with concurrency control in databases one cannot implement such a system by adding one balcony to the other to solve situation-dependent problems. Instead, a proper framework with clear semantics is needed which allows to argue on 'correctness' and which covers all possible cases (no “implementation holes”). In Sect. 5 we described the research and development work in the ADEPT project, which reflects this thinking to a large degree. As pointed out, the ADEPT project attempts to consider most of the challenges described in conjunction with each other. Looking at the total landscape, the support of dynamic changes and scalability issues are probably understood best in the mean-time. Also component-oriented software development, user-interface issues, and the required support of organizational structures (which is a nightmare in this domain) are understood quite well. The work on temporal issues as well as on supporting inter-workflow dependencies (another important aspect we could not address in this paper) is on its way.

Acknowledgements

We want to thank the other members of the ADEPT research group, Thomas Bauer, Clemens Hensinger, and Thomas Strzeletz, we want to thank Rolf Kreienberg, Raphael Mangold, and Thomas Zemmler from the University Women's Hospital, and the numerous students who contributed to the implementation of $ADEPT_{workflow}$.

References

1. Aalst, W. van der: Proc. Workshop on Workflow-Management: Net-based Concepts, Models, Techniques, and Tools. Lisbon, June 1998, Comp. Science Report Nr. 98-07, University of Eindhoven
2. Aalst, W. van der: Generic Workflow Models: How to Handle Dynamic Change and Capture Management Information? Proc. CoopIS'99, Edinburgh, Sept. 1999, pp. 115–126
3. Agostini, A.; De Michelis, G.: Simple Workflow Models. In [1], June 1998, pp. 146–163
4. Bauer, T.; Dadam, P.: A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration. Proc. CoopIS'97, Kiawah Island, June 1997, pp. 99-109
5. Bauer, T.; Dadam, P.: Efficient Distributed Control of Enterprise-Wide and Cross-Enterprise Workflows. In [11], October 1999, pp. 56–64
6. Badouel. E.; Oliver, J.: Reconfigurable Nets, a Class of High Level Petri Nets Supporting Dynamic Changes. In [1], June 1998, pp. 129–145
7. Bogia, D.: Supporting Flexible, Extensible Task Descriptions In and Among Tasks. Ph.D. Thesis, University of Urbana, Illinois, 1995
8. Casati, F.; Ceri, S.; Pernici, B.; Pozzi, G.: Workflow Evolution. Data & Knowledge Engineering, Vol. 24, 1998, pp. 211-238
9. Dadam, P.; Kuhn, K.; Reichert, M.; Beuter, T.; Nathe, M.: ADEPT: An Integrated Approach for the Development of Flexible, Reliable, Cooperative Systems in Clinical Application Environments. Proc. GI-Jahrestagung (GISI 95), Zürich, 1995, pp. 677-686 (in German)
10. Dadam, P.; Reichert, M.; Kuhn, K.: Clinical Workflows – The Killer Application for Process-oriented Information Systems? Ulmer Informatik-Bericht No. 97-16, University of Ulm, November 1997
11. Dadam, P.; Reichert, M. (eds.): Proc. Workshop on Enterprise-Wide and Cross-Enterprise Workflow-Management, 29. Jahrestagung der GI (Informatik'99), Paderborn, Germany, Oct. 1999 (appeared as Ulmer Informatik-Bericht, No. 99-07, University of Ulm)
12. Dechter, R.; Meiri, I.; Pearl, J.: Temporal Constraint Networks. Artificial Intelligence, Vol. 49, 1991, pp. 61-95
13. Dijkstra, E.: Cooperating Sequential Processes. In: Genuys, F. (ed.): Programming Languages, Academic Press, London, 1968
14. Ellis, C.A.; Keddara, K.; Rozenberg, G.: Dynamic Change Within Workflow Systems. Proc. COOCS'95, Milpitas, CA, 1995, pp. 10-21
15. Hagemeyer, J.; Hermann, T.; Just-Hahn, K.; Striemer, R.: Flexibility in Workflow Management Systems. In: Likowski, R. et al. (eds.): Usability Engineering, Teubner, 1997, pp. 179-190
16. Han, Y.: Software Infrastructure for Configurable Workflow Systems. Ph.D. Thesis, TU Berlin, 1997
17. Heinlein, C.; Kuhn, K.; Dadam, P.: Representation of Medical Guidelines Using an Classification-Based System. Proc. CIKM '94, Gaithersburg USA, November 1994, pp. 415-422
18. Jablonski, S.; Stein, K.; Teschke, M.: Experiences in Workflow Management for Scientific Computing. Proc. 8th Int'l Workshop on Database and Expert Systems, Toulouse, 1997, pp. 56-61
19. Joeris, G.: Defining Flexible Workflow Execution Behaviors. In [11], Oct. 1999, pp. 49–55
20. Karbe, B.; Ramsperger, N.; Weiss, P.: Support of Cooperative Work by Electronic Circulation Folders, SIGOIS Bulletin, Vol. 11, 1990, pp. 109-117
21. Klein, M.; Dellaroca, C.; Bernstein, A.: CSCW-98 Workshop: Towards Adaptive Workflow Systems (Held in conjunction with the CSCW Conf.), Seattle, WA, Nov. 1998
22. Kradolfer, M.; Geppert, A.: Dynamic Workflow Schema Evolution Based on Workflow Type Versioning and Workflow Migration. Proc. COOPIS'99, Edinburgh, September 1999

23. Leape, L.L. et al.: The Nature of Adverse Events in Hospitalized Patients. New England Journal of Medicine, 324, 1991, pp. 377-384

24. Leymann, F.; Roller, D.: Workflow-based Applications. IBM Sys Journal, Vol. 36, 1997, pp. 102-123

25. Liu, C.; Orlowska, M.; Li, H.: Automatic Handover in Dynamic Workflow Environments. Proc. CAiSE'98, Pisa, Italy, June 1998, pp. 159–171

26. Liu, L.; Pu, C.: Methodical Restructuring of Complex Workflow Activities. Proc. 14th Int'l Conf. On Data Engineering (ICDE'98), Orlando, Florida, February 1998, pp. 342-350

27. Luo, Z.; Sheth, A. et al.: Defeasible Workflow, its Computation and Exception Handling. In [21], 1998

28. Müller, R.; Rahm, E.: Rule-Based Dynamic Modification of Workflows in a Medical Domain. Proc. Datenbanksysteme in Büro, Technik und Wissenschaft, Freiburg, March 1999, pp. 429-448

29. Reichert, M.; Bauer, T.; Dadam, P.: Enterprise-Wide and Cross-Enterprise Workflow-Management – Challenges and Research Issues for Adaptive Workflows. In [11], Oct. 1999, pp. 56–64

30. Reichert, M.; Dadam, P.: $ADEPT_{flex}$ – Supporting Dynamic Changes of Workflows Without Losing Control. Journal of Intelligent Information Systems, Vol. 10, 1998, pp. 93–129

31. Reichert, M; Hensinger, C.; Dadam, P.: Supporting Adaptive Workflows in Advanced Application Environments. Proc. EDBT Workshop on Workflow Management Systems, Valencia, March 1998, pp. 100-109

32. Weske, M.: Adaptive Workflows based on Flexible Assignment of Workflow Schemas and Workflow Instances. In [11], Oct. 1999, pp. 42–48

33. Workflow Management Coalition: Workflow Management Application Programming Interface (Interface 2 & 3) Specification. Document No. WFMC-TC-1009, Version 2.0, 1998