Preprint of: Buchmann, A.P. (ed.): Proceedings of BTW99, Freiburg im Breisgau, 1.-3. März 1999. Spinger, Berlin 1999: 429-448.

Rule-Based Dynamic Modification of Workflows in a Medical Domain

R. Müller, E. Rahm

Institut für Informatik, Universität Leipzig, Augustusplatz 10/11, 04109 Leipzig
{mueller,rahm}@informatik.uni-leipzig.de

Abstract. A major limitation of current workflow systems is their lack of supporting dynamic workflow modifications. However, this functionality is a major requirement for next-generation systems in order to provide sufficient flexibility to cope with unexpected situations and failures. For example, our experience with data intensive medical domains such as cancer therapy shows that the large number of medical exceptions is hard to manage for domain experts. We therefore have developed a rule-based approach for partially automated management of semantic exceptions during workflow instance execution. When an exception occurs, we automatically determine which running workflow instances w.r.t. which workflow regions are affected, and adjust the control flow. Rules are being used to detect semantic exceptions and to decide which activities have to be dropped or added. For dynamic modification of an affected workflow instance, we provide two algorithms (*drcd*-and *p*-algorithm) which locate appropriate deletion or insertion points and carry out the dynamic change of control flow.

1 Introduction

The need for dynamic workflow changes to manage new situations and unexpected difficulties during the execution of workflow instances has already been identified as one of the central workflow research issues (e.g. [She97, SK98]). However, most of the proposed approaches assume that a *human* expert decides which changes have to be applied. Furthermore, they concentrate on transformation frameworks for dynamic structure changes to achieve consistency of the modified workflow instance w.r.t. control and data flow aspects. In contrast to this, it has not yet been investigated sufficiently whether machine agents can be enabled to *automatically* decide first when a running workflow instance is no longer adequate, and second which changes of the original workflow definition have to be applied to better reflect the new situation.

To address this problem we describe a rule-based approach for the detection of semantic exceptions and for dynamic workflow modification with a focus on medical workflow scenarios. Dynamic workflow changes are triggered by *semantic exceptions* which, in our domain, are medical events such as a newly detected allergy or a critical laboratory value requiring a change in an ongoing treatment plan (workflow instance). This is done in several steps: First, a rule-based system decides whether a medical event for a patient implies that currently planned or processed medical activities are not appropriate anymore, and whether additional activities become necessary. Second, it is derived which running workflow instances are affected. Third, for an affected

workflow instance, the relevant workflow modification region is determined. This region depends on the type of an exception *E* and the "temporal influence" of *E*, so that changes are only applied to relevant areas. Fourth, within the relevant modification region, workflow modifications are processed to adjust the instance w.r.t. the new situation. Fifth, the modified workflow instance is continued.

Our primary focus thus is on (partially) automated dynamic *control flow* modifications of workflow instances. We concentrate on *local* modifications such as the automated deletion, replacement and addition of single workflow activity nodes, while the deletion or construction of complete subgraphs is not yet addressed. Data flow aspects are omitted as well, as they can be viewed, from the semantic point of view, as a more technical consequence of control flow changes. Furthermore, due to space restrictions we do not deal with *fatal* semantic exceptions like the death of a patient which form – from the technical point of view – a relatively trivial type of semantic exceptions (they definitely terminate workflow instances).

The work described here is part of the workflow system HEMATOWORK [MHL+98, MH98], which supports long-term therapy in the domain of distributed cancer therapy and is currently under development at the University of Leipzig. This domain is suitable for workflow management, as the treatment is mainly based on standardized treatment plans. However, as patients react very different to the aggressive long-term treatment, a significant number of treatments has to be adjusted partially and in a patient-specific manner. By automatically adjusting workflow instances we expect our approach to support physicians because the need for time-consuming manual interactions with the workflow system is reduced. Moreover, the probability to oversee an important semantic exception can be reduced as well. Certainly, *automated* workflow modification will be possible only up to a certain degree as events and their implications may become arbitrarily complex. But even if the entire implications of an event cannot completely be derived by a machine agent, a *partially* automated management is already helpful. In particular, it is already helpful to inform the user which workflow instances w.r.t. which control flow areas are affected by a semantic exception, even if the modifications themselves have to be determined manually. Furthermore, the approach is orthogonal to the question at which step during automated modification the decision should be confirmed by a human expert. For example, the insertion of additional drugs certainly should be confirmed by a physician, while additional diagnostic examinations may be inserted without confirmation. While we focus on medical applications, our approach principally is applicable to other domains as well. In particular, the basic algorithms and agents do not make assumptions about the domain-specific structure of events and exceptions.

The paper is organized as follows. Section 2 discusses related work in the context of dynamic workflows. Section 3 describes the subset of Frame Logic and Transaction Logic [KLW95, BK94] used for the representation of events, activities and rules. We use a logic-based approach as the automation of the dynamic modification process requires formal semantics. F-logic has been selected because it provides object-oriented modeling capabilities combined with deductive aspects. Transaction Logic is needed for formal description of database updates. In Section 4 our basic notations of medical events and activities are formalized. Section 5 contains the model of workflow instance execution and event-caused instance interruptions. Section 6 then

describes the automated workflow modification system with the *drcd*-and *p*-algorithms. These two algorithms are responsible for locating appropriate deletion or insertion points and for carrying out the dynamic change. Section 7 completes the paper with a discussion of the strengths and limitations of the approach, implementation aspects and future work to be done.

2 Related Work

Although a lot of application domains require increased workflow flexibility, this functionality currently is not provided by most commercial workflow systems. Therefore, the development of dynamic workflow systems has been addressed by several researchers during the last years. An overview and classification of dynamic change approaches can be found, for example, in [She97, SK98].

In [RD98], Reichert and Dadam suggest an elaborated workflow execution model and workflow transformation calculus ($\text{ADEPT}_{\text{FLEX}}$) which is able to modify the control and data flow at run-time while maintaining correctness and consistency. Related to this, the WASA approach [VW98, Wes98] provides a set of operators which are able to change the control flow w.r.t. activity instances, such as *SkipInstance(i)* or *RepeatInstance(i*). However, mechanisms that could enable an automated agent to decide when and what structural change should be applied to a workflow instance are not discussed in both cases.

In the MOBILE system [BJ96, HSS96], (sub-)workflow definitions can be left incomplete at pre-defined points, if a final decision about an appropriate subworkflow logic can only be determined at run-time. An incomplete definition is described in terms of goals and partially defined process patterns. However, methods to automatically derive the workflow completion, when the control flow arrives at this point and relevant data are available, are not discussed by the authors.

Within the WIDE workflow model and architecture, Casati et al. [CGP+97] describe a typology of workflow exceptions and an exception handling approach which is mainly based on *condition-reaction* pairs; if a condition is violated and an exception raised, an informative (informing an agent) or a corrective reaction is triggered. The latter for instance may be of type CANCEL_TASK or CANCEL_CASE, or, in general, by aborting or redirecting the control flow. However, the authors do not discuss any reaction sequences which are able to modify the running instance through structural changes.

In [MH98], we sketch an approach to modify workflow instances in a knowledge-based manner. Unfortunately, this approach requires meta information within the workflow definitions indicating at which nodes dynamic modification have to be processed. Furthermore, the approach basically is restricted to dynamic workflow *hierarchical refinement*, and does not support dynamic changes in general.

In summary, to our knowledge, strategies to automatically identify semantic exceptions and change a workflow instance have not yet been investigated sufficiently.

3 Representation Methods

For the purposes of this paper, we use a combination of F-Logic and Transaction Logic (*TL*) [KLW95, BK94, BK95] to formalize the interaction between semantic exception events, rules and workflow instances. F-Logic is used to represent both the data model for medical events and activities, and rules expressing dependencies between events and activities in a natural object-oriented manner. As F-Logic is a pure retrieval-oriented language and does not support database updates, we additionally use several constructs of *TL* to express, for example, the generation of new medical orders when semantic exceptions occur[1].

F-Logic class definitions are of the form:

person [name => string, birthday => date], patient::person, physician::person,
patient [diagnosis => string, responsible-doctor => physician],
physician [degree => {trainee, assistant, senior}, patients =>> patient]

where "=>>" arcs denote *set-valued* attributes, and "::" is the *is-a* operator. Object instances are expressed, for example, via:

Bob:patient [name → „Miller, Bob", birthday → 12/5/1967,
diagnosis → „Leukemia", responsible-doctor → Steve]
Steve:physician [name → "Taylor, Steve", ..., patients →>{Bob, Fred, Mary}]

where ":" is the *is-instance-of* operator. Alternatively, we use "." expressions like *bob.responsible-doctor.degree[d]* for *bob[responsible-doctor → D [degree→ d]]*.

Queries and rules are expressed with the *?-* and ← operator, e.g. (in compact "." expression notation):

?- P:patient ∧ P.diagnosis["lymphoma"] ∧ P.responsible-doctor.name["Taylor, Steve"] // find patients with lymphoma where responsible doctor is Taylor, Steve

D.name["Miller, John"] ← P:patient ∧ P.diagnosis["leukemia type AML"] ∧ P.responsible-doctor[D]
// if patient suffers from special leukemia of type AML, then Miller, John is the
// doctor of this patient (as, for example, he is the AML-specialist in the hospital)

From *TL*, we use the standard update operators *del* and *ins*, which can be applied to every database predicate, e.g. *patients.ins (patient-oid)* (with *patients* being the predicate extension of the *patient* class above). *del* and *ins* have a formal semantics (in contrary, for example, to Prolog's *assert* and *retract*), and any sequence involving database updates can be specified as a transaction via the serial conjunction operator ⊗. This means if any step of the ⊗–sequence fails, the sequence is rolled back.

[1] Note that we do *not* use *TL* to model the workflows themselves (as, for example, in [DKR+98]).

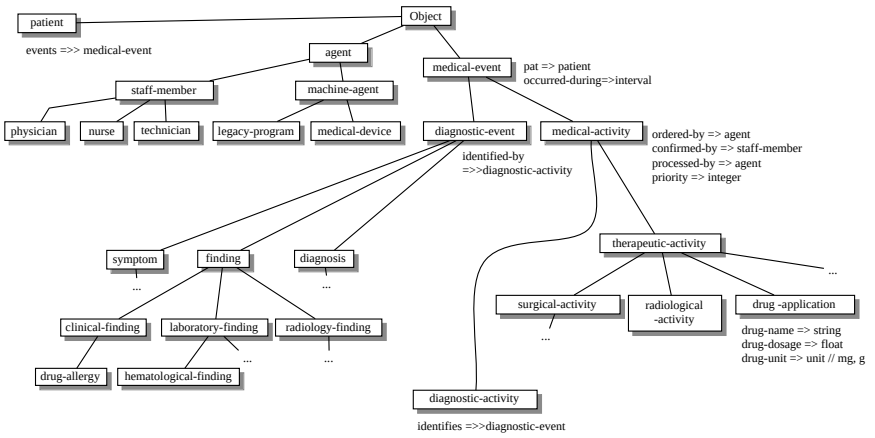

Figure 1: General F-Logic class hierarchy for agents, events and activities in a medical domain. A *legacy-program* is a machine agent which may not allow to be compensated or aborted in a controlled way (e.g. w.r.t. a 2PC protocol). Dots indicate (omitted) subhierarchies for particular subdisciplines.

Further *TL*-operators such as the | operator for concurrent execution are not needed here as we do not use *TL* for workflow definition.

4 Medical Events and Workflows

In the following, we formalize our notation of medical events, activities, semantic exceptions, and workflows over medical activities.

4.1 Medical Events, Activities, and Semantic Exceptions

Fig. 1 shows the F-Logic data model hierarchy of event and activity classes we use in this paper. The class *medical-event* is the abstract super class of everything that "happens" to a patient and that has a medical relevance. The *medical-event* subclass *diagnostic-event* is the root class for all data describing diagnostic information about the patient, while the subtree starting from *medical-activity* provides classes for the storage for all medical activities done with the patient (e.g. drug applications). For the purposes of this paper, we assume in a patient database *DB* the two associated basic predicates:

diagnostic-events (D) with *D : diagnostic-event*

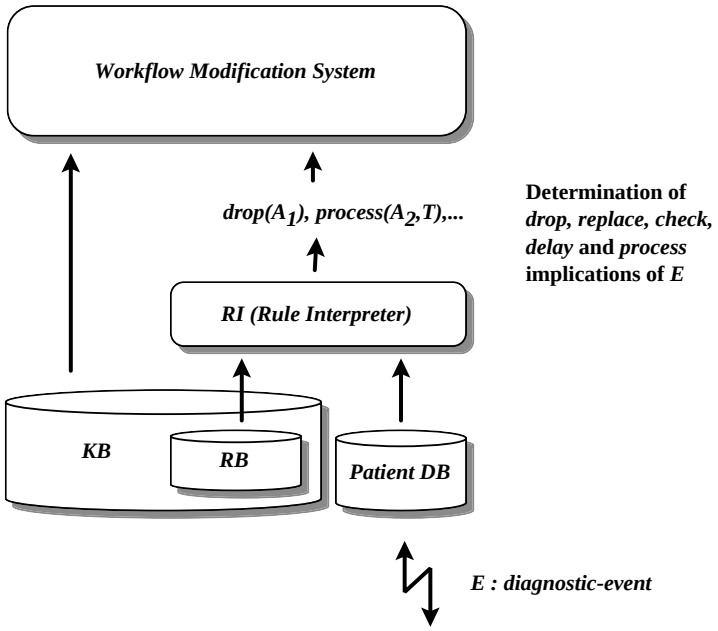

Figure 2: Basic components for the detection of semantic exception events and the derivation of implications for medical activities. The modification system is described in sec. 6.

which is the object extension storing all diagnostic events that have been observed or detected w.r.t. the patients, and

activities-ordered (A) with *A : medical-activity*

which records all activities that already have been ordered or suggested by human or automated agents (but may not yet have been carried out).

A *semantic exception* is a *diagnostic-event* instance which implies that the current treatment has to be modified (e.g. because of an allergy). The interpretation whether or not an event constitutes a semantic exception is the task of a rule base *RB* and a rule interpreter *RI*. Furthermore, we assume a knowledge base *KB* incorporating *RB* and containing patient-independent information about medical events, such as the duration and dosage of a specific drug application (see fig. 2).

4.2 Derived Predicates on Activities

As *RI* derives statements about medical activities, we now introduce predicates on *medical-activity* instances. Therefore, let *Pred = {drop, replace, check, delay, process}* be our set of *activity* predicate symbols, *A* an object of class *medical-activity*,

Rule 1:

drop(D)
←
L : hematological-finding[pat → P, parameter → LEUKOCYTE-COUNT, unit → #/mm³, value → V] ⊗ V < 1500 ⊗ D:drug-application[pat → P', drug-name → „ETOPOSID"] ⊗ P' == P ⊗ activities-ordered.del(D).

Rule 2:

process(D, [0, 2 days]) // give drug D within two days
←
L:hematological-finding[pat → P, parameter → LEUKOCYTE-COUNT, unit → #/mm³, value → V] ⊗ V < 2000 ⊗ D:drug-application ⊗ new-id(D) ⊗ D.init[pat → P, drug-name → „G-CSF", drug-dosage → 300, drug-unit → µg] ⊗ activities-ordered.ins(D)

Figure 3: Sample rules deriving implications for medical activities.

and *T* a temporal constraint. *T*, for example, may specify the interval during which an activity should be processed. Then we define with

drop(A)	*// drop activity A*
replace(A_1,A_2)	*// replace A_1 with A_2*
check(A)	*// check whether activity A is still appropriate*
delay (A,T)	*// delay A w.r.t. temporal constraint specified by T*
process(A,T)	*// process A within the temporal constraint specified by T*
	// (e.g. apply drug D within the next two days)

the associated predicate molecules. The first four predicates deal with *semantic failures* of activities. *drop* and *replace* express "hard" failures (with the consequence *"do not process activity anymore"*), while *check* and *delay* address "weak" failures. Fig. 3 contains some sample rules concerning implications of semantic exceptions. Rule 1 orders the discontinuation of the drug ETOPOSID when the leukocyte count is less than 1500 per mm^3, while rule 2 suggests the administration of the so-called *GRANULOCYTE COLONY STIMULATING FACTOR (G-CSF)* for hematological recovery when the leukocyte count is less than 2000.

The main task of the modification system we will describe in sec. 6 is to translate derived predicates on activities into workflow instance modifications. However, we first briefly describe our workflow representation and execution model.

4.3 Workflow

In our context, we view a workflow as a control flow over medical activities. A workflow node *n* represents exactly one *activity* (e.g. a drug application or a x-ray

examination) which is processed when the control flow reaches *n*.[2] For workflow build-time purposes, we call a construct of the form

class-name[attr$_1$ $\rightarrow\!\!\!\!\!\!^{3}$ *val*$_1$*, ..., attr*$_n$ $\rightarrow$ *val*$_n$*]* with *class-name::medical-activity,*
e.g.
drug-application[drug-name$\rightarrow$*"CISPLATIN",drug-dosage*$\rightarrow$*200, drug-unit*$\rightarrow$ *mg]*

a *medical-activity instance definition.* Such an instance definition can be assigned to a node to indicate that the node at run-time manages (i.e. processes) an activity instance with this properties (the definition is called *partial* if some attribute values are not specified at build-time but will be determined at run-time[4]).

Let *Activities* be a set of (partial) activity instance definitions. A workflow *W* over *Activities* is a 6-tuple *W=(Nodes*$_W$ *, NAM*$_W$ *, Edges*$_W$ *, ETM*$_W$ *, ECM*$_W$ *, SemSeq*$_W$*)* where

- *Nodes*$_W$ is a set of workflow nodes with *Nodes*$_W$ = *Activity-Nodes*$_W$ $\cup$ *Control-Nodes*$_W$. While *Activity-Nodes*$_W$ is the set of *activity nodes*, *Control-Nodes*$_W$ contains the *control flow nodes* of types START, END, OR-SPLIT, AND-SPLIT, OR-JOIN, LOOP-START, or LOOP-END.
- the function *NAM*$_W$ *: Activity-Nodes* $\rightarrow$ *Activities* maps (partial) activity object instance definitions to the activity nodes.
- *Edges*$_W$ $\subset$ *Nodes*$_W$ $\times$ *Nodes*$_W$ is the set of directed (control flow) edges. The subset *Edges*$_{W,sync}$ $\subset$ *Edges*$_W$ contains *synchronization* edges (see, for example, [RD98,ASE+96]). The semantics of a s*ynchronization* edge *e = (n,m)* is that *m* may only be activated when *n* has completed, or when *n* is not reachable anymore by the control flow[5].
- *ETM*$_W$ *: Edges*$_W$ $\rightarrow$ *TemporalObjects* assigns a temporal object such as an exact duration (e.g. "4 days") or an interval to an edge to indicate temporal transition conditions such as *"Next chemotherapy starts exactly 4 days after previous one"*. By convention, *ETM*$_W$ maps the temporal NULL object to an edge when no temporal transition condition exists.
- *ECM*$_W$ *: Edges*$_W$ $\rightarrow$ *Conditions* assigns a F-Logic condition such as *"... [parameter* $\rightarrow$ *LEUKOCYTE-COUNT, value* $\rightarrow$ *V]* $\wedge$ *V < 1500"* to an edge, which must be fulfilled when the control flow wants to pass an edge. Again, by convention, we set *ECM*$_W$*(e)* = NULL when there is no transition condition.

---

[2] We could also assign several activities to a node. However, we favor a 1:1 relationship between nodes and activities, as this facilitates and unifies workflow modification such as dropping an activity.

[3] or $\rightarrow\!\!>$ for set-valued attributes

[4] E.g. the particular dosage of a drug may not be determined before node execution.

[5] See [RD98] for a more detailed typology of synchronization edges. For the scope of this paper, however, we restrict these edges to the "standard" type described above.

- $SemSeq_W \subset \{ \{k_1,..,k_m\} \subset Activity\text{-}Nodes_W \mid k_1,..,k_m$ *is sequence within W}* is the set of activity sequences that should not be broken up by inserting additional activity nodes.

Remarks: The partition of transition conditions into temporal and "non-temporal" conditions is mainly motivated by the introduction of so-called *interval covering workflow regions* in sec. 5, which need ETM_W for determination. The problem that a "non-temporal" transition condition such as *"LEUKOCYTE-COUNT < 1500"* implicitly has a temporal aspect in the meaning of *"Wait the number of days until LEUKOCYTE-COUNT < 1500"* is discussed in this sec. too.

$SemSeq_W$ denotes meta information for the modification system w.r.t. the question which sequences – from the application view – form semantic and "atomic" units (e.g. *Perform Chemotherapy Toxicity Test* $\rightarrow$ *Prepare Patient for Chemotherapy* $\rightarrow$ *Perform Chemotherapy*) and should not be broken up by inserting additional activities. In this case, the modification agent, for example, tries to insert a parallel branch for the additional activity. $s \in SemSeq_W$ can be user-defined or constructed by the *inter-activity-dependency* predicate (see sections 5 and 6).

$Activities_W$ denotes the set of all *medical-activity* instance definitions used by *W*. Furthermore, to reduce formal complexity, we do not consider *nested* workflow definitions (i.e. workflows with abstract aggregated nodes representing subworkflows). If nested workflows are defined at build-time, we assume that such a workflow definition is transformed at run-time into a flat workflow by replacing the aggregation nodes by their subworkflows.

5 Workflow Execution and Interruption

We now formalize our execution model and notion of an event-caused instance interruption: With i_W, we denote a run-time instance of the definition *W*, and assign a control agent CA_{i_W} to every i_W, which controls and processes the instance. Furthermore, CA_{i_W} also is responsible for dynamically modifying the control flow of i_W (what will be described in sec. 6). To every instance i_W, we can assign exactly one object *P : patient* treated by i_W. This assignment is expressed by the function *get-patient*(i_W).

When i_W is generated by the engine, the following initializing routine is processed:

> ***for*** $a \in Activities_W$ *: // a is used by W*
> *A:medical-activity* $\otimes$ *new-id(A)* $\otimes$ *copy-values(A,a)* $\otimes$ *A.pat[get-patient*(i_W)*]* $\otimes$ *activities-ordered.ins(A)*

where *new-id* is a *TL* predicate binding *A* to a new oid each time this predicate is evaluated[6], and *copy-values* a function that copies the values of the partial instance definition *a* to the attributes of *A*. This is necessary as the automated modification

[6] see [KLW95], sec. 17.4, for details concerning object generation in F-Logic/TL

system of sec. 6 has to reason about the activities the instance is currently processing or will perhaps process depending on the control flow. [7]

At the occurrence of an event E during execution, an activity node $n \in$ *Activity-Nodes* has a state described by the function $state_E(n) \in$ *{untouched, activated, currently-processed, committed, failed},* where

- *untouched* means that the control flow has not yet reached node n (or will never reach n at all w.r.t. the particular workflow instance i_W).
- *activated* means that control flow has reached the input edges of n, and that the conditions *ETM(e)* and *ECM(e)* are fulfilled for every input edge e.
- *currently-processed* means that n is currently processed by CA_{i_W}.
- *committed* means that the processing of n has been successfully completed.
- *failed* means that n is not appropriate anymore. Each of the states *untouched, activated, currently-processed* may change directly to *failed.* The relationship of the *failed* state with the failure predicates *drop, replace, check* and *delay* of sec. 4.2 is that after *RI* has derived, for example, a *drop(A)* statement for an activity A, it is automatically explored which workflow nodes w.r.t. which workflow instances are affected. These nodes are then temporarily set to state *failed,* and then replaced or dropped by the dynamic modification system (see sec. 6). *Technical* failures of steps (e.g. because of an engine crash) are beyond the scope of this paper.

5.1 Exception-Caused Workflow Interruption

We say that i_W has been E-interrupted at node set $N_E=\{n_1,..,n_k\}$, if CA_{i_W} has interrupted the control flow of i_W because of the occurrence of an event E, and if – at the moment of the interruption – $state_E(n_i)$ is *activated* or *currently-processed* for all n_i. The cardinality of N_E is > 1, if i_W is interrupted within parallel execution (i.e. after an AND-SPLIT).

5.2 Exception-Related Workflow Modification Regions

An important subproblem of dynamic workflow modification is the question *for which workflow region it is "sensible" at all to reason about structural changes* when a semantic exception occurs. For example, if it is detected during a chemotherapy that a patient has an *allergy* w.r.t. one of the drugs, it makes sense to delete *every* node in the workflow instance which applies this drug (as it is too dangerous to administer this drug furthermore). However, if the patient only shows a *moderate toxicity* w.r.t. an important drug, it makes sense to delete the drug nodes only in a „nearer" neighborhood, and to assume that in workflow areas „far away" the drug will be applied again, as patients usually recover from moderate toxicity effects and are then able to take this drug again after a while. Furthermore, in domains where semantic

[7] We omit techniques to iteratively and on-demand generate the *medical-activity* objects during instance execution, as this is a performance aspect.

exceptions occur frequently, it is useless to reason about workflow modifications which would affect regions which are – w.r.t. the number of activities – „further away“ from the current control flow, as another exception may happen in the meantime creating a new situation. The basic problem of modification regions is, of course, that they hardly can be specified at build time, but depend on the type of the exception event *E*.

For our purposes, we formalize a modification region (*MR*) as follows: Given a workflow *W*, an event *E* and an interruption node set $N_E = \{n_1,..,n_k\} \subset Nodes_W$, a *MR* r_{W,E,N_E} syntactically is a complete subgraph of *W* containing N_E and having the *strict-forward* characteristic. The latter means, that for all $m \in r_{W,E,N_E}$, there exists a $n_m \in N_E$ and a path $n_m \rightarrow k_1 \rightarrow ,...,\rightarrow k_j \rightarrow m$ with nodes from r_{W,E,N_E} (i.e. the control flow starting at n_m may reach *m* during execution)[8].

The determination of a *MR* r_{W,E,N_E} is done in three basic steps: First the interval *T* expressing the “temporal influence” of *E* is determined. Second, for every path starting from N_E a subpath covering *T* is determined. Third, r_{W,E,N_E} is constructed by the union of this subpaths. These steps are now described in detail:

Temporal influence of events: For this, we have to assume some heuristic knowledge about the „temporal scope“ of events. We therefore assume in our medical knowledge base *KB* patient-*independent* information about events (similar to the average duration of medical activities above), such as the definition of drug-caused hematological toxicity and the average time a patient needs to overcome such a drug-caused side effect (although this is complex knowledge, it is often already available in current medical knowledge-based systems; see, for example, [MSN+97, MTD+96]). The *KB* predicate for *MR* determination is *kb-temporal-scope (kb-E, T)*, where *kb-E* is a (partial) instance definition of a *KB* class hierarchy, and *T* an interval information. *kb-temporal-scope* represents the usual or average time period during which an event with the characteristics of *kb-E* affects the activities processed for an usual patient. For example, for a drug allergy we would have *kb-temporal-scope (kb-drug-allergy[], [0,∞])* to represent that a detected drug allergy influences the therapy of the patient for the *whole* remaining treatment (whatever its particular characteristics are). In opposite to this, for a moderate hematological side-effect such as a decreased leukocyte count as a consequence of aggressive drugs, we would have

kb-temporal-scope (kb-hematological-finding
[type → CYTOSTATIC-DRUG-CAUSED-HEMATO-TOXICITY,
leukocyte-interval → [500 #/mm^3, 2500 #/mm^3],
[0,2 weeks])

[8] Non-strict forward-oriented *MRs* make sense, when we have to consider that the workflow is rolled back and reset to a node from which additional nodes become reachable. The construction of such roll-back insensitive *MRs* may be appropriate for dangerous events (such as a serious hematological toxicity w.r.t. a drug *D*) so that in case of a roll-back it is guaranteed that *D* will also not be applied. Because of limited space, we do not address this type of *MRs*.

to represent that such an event influences the workflows only for a period of about 2 weeks (during which the responsible drugs should be dropped), as it can be assumed that the patient will recover during these two weeks and will be able to take the drugs again[9].

Interval covering region: Given a workflow definition *W*, a node *n* and an interval *T* of type *[0, k time-units]*, the *T*-covering region $r_{cover,W,n,T}$ is a strict forward-oriented region that covers those nodes that will – in the normal case (i.e. if *no* additional semantic or technical exceptions occur) – be processed during the interval *T* (beginning with *n*). The determination of a *T*-covering region $r_{cover,W,n,T}$ is achieved as follows: For every path p_n starting from *n* the following is done: for the direct successor activity node *m* of *n* (w.r.t. p_n) heuristic information in the knowledge base *KB* about the normal duration of the associated activity is retrieved. Then, the transition time w.r.t. the successor of *m* is determined (by using the function ETM_W). This is done iteratively until the subpath $p'_n \subset p_n$ is found with the characteristic that the subpath *(p'_n + next successor)* would exceed the duration specified by *T*. $r_{cover,W,n,T}$ is then constructed by the union of all of these subpaths p'_n originating from *n*.

Remark: The determination of $r_{cover,W,n,T}$ faces two problems: First, if for an edge *e* of a path p_n it holds that ECM_W *!=* NULL, this means that the transition duration is unpredictable at determination time of $r_{cover,W,n,T}$. At the moment, we consider such an *e* simply with a duration of *max(0,ETM(e))*, with the consequence that $r_{cover,W,n,T}$ may contain nodes that are executed *after* the interval *T*. In other words, the nodes of $r_{cover,W,n,T}$ in this case form a superset of the nodes executed during *T*.[10] Second, if a p_n contains a loop with a termination condition such as *Do Chemotherapy activities until x-ray cannot detect any tumor remainder anymore* the execution time of the loop is unpredictable too. Again, without additional meta knowledge we can only assign a default duration such as the duration of its activity sequence to this loop, with the consequence that $r_{cover,W,n,T}$ again becomes a *non-minimal T*-covering region. In both cases, $r_{cover,W,n,T}$ only is "close" to the region which will in fact be executed during *T*. However, this is still better than changing the workflow w.r.t. regions which are not relevant at all.

Final construction: The *MR* r_{W,E,N_E} is then constructed by the union of all $r_{cover,W,n,T}$ for $n \in N_E$.

---

[9] If not, the rule system would derive another semantic exceptions after this period stating that the patient shows an exceptional recovery pattern that requires additional procedures.

[10] The determination of the minimal $r_{cover,W,n,T}$ in such cases would require additional meta knowledge such as the average time $t_{C,lastValue}$ it takes for a patient after a chemotherapy of type *C* to increase the LEUKOCYTE-COUNT from *lastValue* to *1500*. As there exist detailed biomathematical models for cell colony growing in the oncological domain, we can assume this knowledge, for example, for HEMATOWORK.

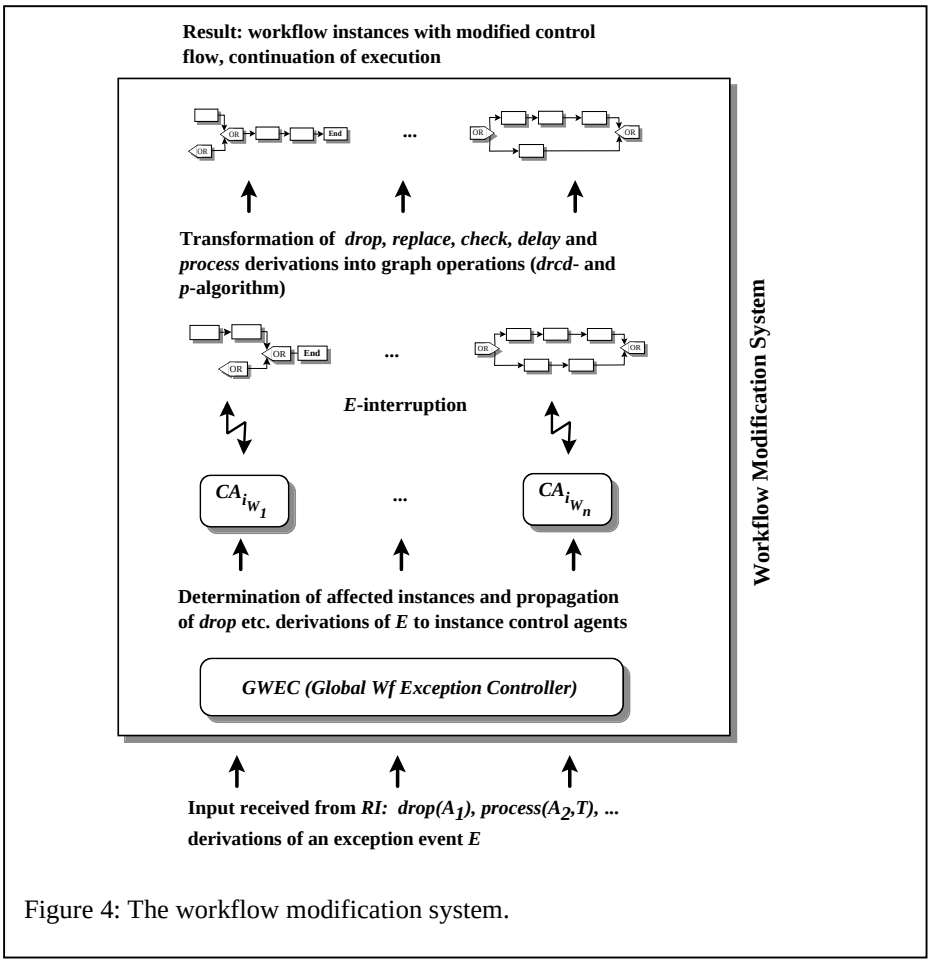

Figure 4: The workflow modification system.

6 The Dynamic Workflow Modification System

We now describe the dynamic workflow modification system in detail (see fig. 4 for an overview). We therefore assume that *E* is a new event inserted into *diagnostic-events(D)* in the patient *DB* (of fig. 2). This triggers the rule interpreter *RI*, which fires a set of rules R_1 ,..., R_n. For the sample rules of fig. 3, the event

E : hematological-finding[parameter → *LEUKOCYTE-COUNT ,..,value* → *1200]*

would induce rules 1 and 2, which would derive that the drug *ETOPOSID* should be dropped and that for hematological recovery an additional drug, the so-called *G-CSF (GRANULOCYTE COLONY STIMULATING FACTOR)*, should be given.

RI then passes the set of all derived predicates $dp = \{drop(A_1),.., replace(A_m , A_{m+1}) ,.., delay(A_n ,..) ,.., check(A_o) ,.., process(A_p ,..) ,..\}$ [11] to the global workflow exception controller *GWEC*. This agent, which constitutes the first layer of the modification system, identifies the set of possibly affected instances $pai_E=\{i_{W_1} ,..,i_{W_n}\}$ for the respective patient (by checking whether $get\text{-}patient(i_W)=E.pat$ holds). The further modification steps depend on the activity predicate type and are described in sections 6.1 and 6.2.

An important subproblem w.r.t. all predicate types is to determine whether a modified workflow definition leads to new inter-activity dependencies which have to be expressed. For example, if it is necessary to *insert* a new activity node, it must be decided whether this node can be directly executed „after" the interruption location, or whether inter-activity constraints do require that it has to be executed somewhere later on. For instance, if it is has been derived that a drug D_1 should be given additionally, but that D_1 should never be applied directly before a drug D_2 (as D_2 may neutralize or weaken the effects of D_1), then the node representing the application of D_1 must be executed *after* the D_2 node.

To handle this problem of inter-activity constraints, we assume in our knowledge base *KB* information about dependencies w.r.t. the temporal order or the exclusion of medical activities. In particular, we assume a predicate of the form

$$inter\text{-}activity\text{-}dependency(A_1,A_2,Type,T)$$

where A_i is a medical-activity instance definition (see sec. 4.3). *Type* may, for example, have the values *BEFORE* or *EXCLUSION*, and *T* is an (optional) temporal object such as an interval. For example, the predicate

inter-activity-dependency(drug-application[drug-name→ "VINDESIN"],
drug-application[drug-name→ "CISPLATIN"],
BEFORE, 1 hour)

expresses that the drug *VINDESIN* must be given exactly one hour before the drug *CISPLATIN*. This knowledge type is already incorporated in many medical knowledge-based systems such as the one of HEMATOWORK.

6.1 Management of *drop, replace, check* and *delay* derivations (*drcd*-algorithm)

To identify the set of in fact affected instances $ai_{E,\ drop,\ replace,\ check,\ delay}$, it is first checked by *GWEC* for every $i_W \in pai_E$, whether an activity A_k , for which *drop, replace, check* or *delay* has been derived, matches at least one entry of $Activities_W$.

For $i_W \in ai_{E,drop,\ replace,\ check,\ delay}$, *GWEC* then sends an interrupt request to CA_{i_W} , which performs an *E*-interrupt of i_W at a node set N_E. CA_{i_W} now determines the *E*-

[11] We assume that the knowledge base itself has checked that its knowledge is consistent in the meaning that, given any event constellation, it cannot be derived that an action *A* has to be processed *and* has to be aborted simultaneously. This can achieved by the usual knowledge verification strategies (e.g. [SSS82]).

related *MR* $r_{W,\,E,\,N_E}$ and − within $r_{W,\,E,\,N_E}$ − the set of affected activity nodes by using the function NAM_W. In the example of fig. 5, where the large rectangle represents the *MR* for the considered event *E*, it would be detected that the derived predicate of rule 1 in fig. 3 ("drop drug with name *ETOPOSID*") affects node l_1, as its activity instance definition represents the application of this drug. Depending on the particular predicate type (*drop, replace* etc.), graph operations are then started to, for example, eleminate nodes from the control flow (as the node l_1 in fig. 5). In the case of a *replace* predicate (new node!) or *delay* predicate (existing node at new location!), CA_{i_W} furthermore inspects the *inter-activity-dependency*$(A_1,A_2,Type,T)$ predicate in *kb* to check if there are dependencies with other existing activities. A *BEFORE* dependency, for example, is translated into a synchronization edge which is inserted into the workflow graph as well.

6.2 Management of *process* derivations (*p*-algorithm)

The insertion of *new* nodes (because of a derived *process*$(A_{new}\ ,T)$ predicate) is different from the dropping or replacing of nodes, as we first have to identify *where the node should be placed at all.*

The brute method simply could be to generate a *new* workflow instance with the new activity as the only node. However, this strategy is not appropriate in scenarios where semantic exceptions occur frequently (such as in HEMATOWORK). First, workflow management overhead would be a consequence, because the system must initialize, register etc. a lot of small instances. Second, dependencies between the new activity and activities of already running instances would be difficult to manage.

If we therefore want to insert the new activity into existing instances, we have to decide *which* instance i_W with *get-patient*(i_W)*=E.pat* should take over the task of performing this new activity. This problem certainly cannot be solved in general, as it depends on the particular structure of the workflow system and application domain. However, we can identify the following possibilities of a semi-automated identification of an appropriate i_W:

- **Identification by meta knowledge**: This means that *GWEC* can access basic meta information about the semantic scope of an i_W. For example, in HEMATOWORK, for every treated patient there are two workflow instances, one for the therapeutic activities and one for monitoring tasks for the parallel diagnostic control. Therefore, if the derived activity would be of type *therapeutic-activity*, then HEMATOWORK's *GWEC* would select the therapeutic instance.
- **Using *Activities*$_W$**: If the provided high-level meta knowledge is not sufficient, we can directly use the set *Activities*$_W$, which contains the activity instance definitions referenced by *W* (sec. 4.3). This set, however, may contain an arbitrary mixture of diagnostic and therapeutic activities. Therefore, we can only use aggregation strategies such as determining the cardinality of the subset of diagnostic or therapeutic activities, to identify if *W* is a "more" diagnostic or therapeutic workflow, and to derive some statements about the semantic scope of *W*.

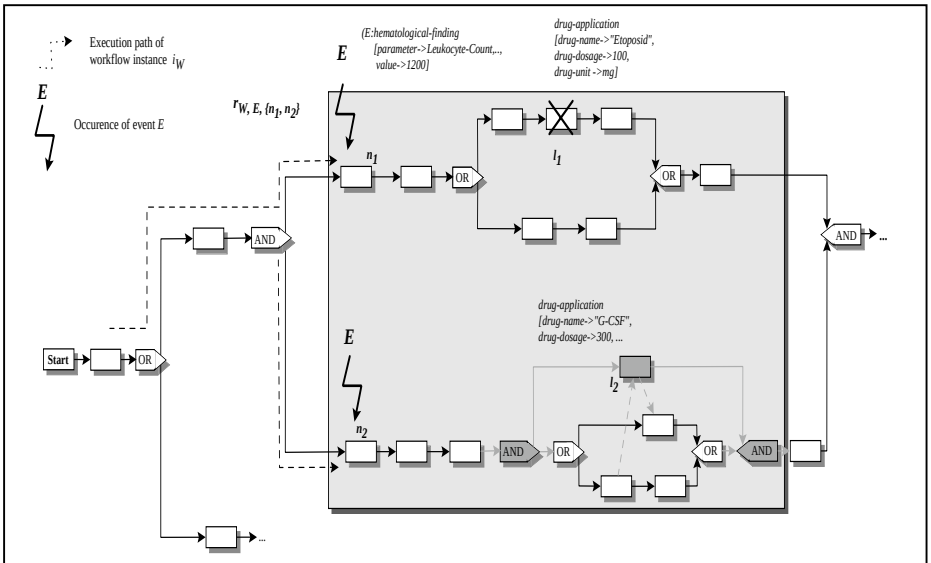

Figure 5: Dynamic node deletion and insertion after an instance interruption caused by exception event *E*. The large rectangle $r_{W, E, \{n_1,n_2\}}$ is the modification region (*MR*), the gray nodes (e.g. $l_2 = A_{new}$) and edges have been inserted because of a *process* predicate (the dashed edges denote synchronization edges).

- **User-driven selection:** If an automatic identification is not possible, the domain expert has to select the appropriate i_W. This can be used in combination with the two strategies described before (e.g. automated suggestion, confirmation by user).

If an i_W has been identified for the new activity A_{new} and then *E*-interrupted, it has to be decided how the A_{new}-node has to be inserted.

Principally, there are two ways to integrate A_{new} into *MR*: It can be inserted into an existing sequence (1), or we can add an AND-SPLIT/AND-JOIN-sphere, which contains (only) the A_{new}-node as one parallel branch, and the already existing "local" nodes as the other parallel branch (2). Possibility (1), for example, is not appropriate, when a sequence $A_1 \rightarrow ... \rightarrow A_n$ being a candidate for insertion of A_{new} forms a *logical unit* (i.e. is an element in $SemSeq_W$) that should not be "destroyed" by inserting other activities. If the sequence $A_1 \rightarrow ... \rightarrow A_n$ allows the integration of A_{new}, we have to consider ordering constraints stored in the *inter-activity-dependency* predicate, and perhaps have to reorganize the sequence. Otherwise, if an AND-sphere is inserted, ordering constraints have to be managed via the additional insertion of synchronization edges. In particular, the *p*-algorithm (*pa*) works as follows: (see [Mül99] for the full algebraic notation):

After the *E*-interrupt of i_W at node set N_E, *pa* first selects a $n \in N_E$ as starting point[12]. It then determines the relevant modification region *MR* which is the intersection of the *E*-related *MR* r_{W, E, N_E} and the *T*-covering region $r_{cover,W,n,T}$. Within *MR*, *pa* determines the sequence $n=k_1 ,.., k_m= c$, where *c* is the next control node of type AND-SPLIT/OR-SPLIT (or the last node of the sequence within *MR*). If $k_2 \rightarrow ,.., \rightarrow k_{m-1}$ is not an element in $SemSeq_W$, A_{new} is inserted into this sequence in a way not violating any constraints in *inter-activity-dependency(...)*. If $(k_2 \rightarrow ,.., \rightarrow k_{m-1}) \in SemSeq_W$, or no constraint-compliant sequence $k_2 \rightarrow ,.., A_{new} ,.., \rightarrow k_{m-1}$ is possible, *pa* alternatively inserts an AND-SPLIT after *n* and an AND-JOIN before *c*, and inserts AND-SPLIT $\rightarrow A_{new} \rightarrow$ AND-JOIN as one branch, and AND-SPLIT $\rightarrow k_2 \rightarrow ,.., \rightarrow k_{m-1} \rightarrow$ AND-JOIN as the other path. However, if an inter-activity dependency of type *EXCLUSION* with one k_i does not allow the insertion of A_{new} in the $(k_2 \rightarrow ,.., \rightarrow k_{m-1})$-context at all, *pa* moves to the *c*-node. If *c* is an AND-SPLIT, *pa* tries to insert A_{new} into one of the parallel sequences after *c* by the same criteria described above. In case of *c* is an OR-SPLIT, *pa* tries to directly insert an AND-SPLIT before *c* and an AND-JOIN after the corresponding OR-JOIN, with AND-SPLIT $\rightarrow A_{new} \rightarrow$ AND-JOIN as one parallel path and the OR-sphere as the other parallel path (see fig. 5). This moving to the next SPLIT-control is done iteratively until A_{new} has been inserted, or until *pa* leaves *MR*. In the latter case, additional user input is required specifying, for example, that a sequence in *MR* is not anymore in $SemSeq_W$, and therefore can be used for the insertion of A_{new}.

After A_{new} has been successfully inserted, the next step now is to determine inter-activity dependencies w.r.t. already existing activities. This is done by scanning *inter-activity-dependency(A,B,Type,T)* in *kb*, and translating them, in the case of a *BEFORE* dependency, into a synchronization edge. In fig. 5, for example, $A_{new} = l_2$ (a G-CSF-node) is synchronized with two other nodes.

7 Summary, Discussion and Future Work

We described an approach for automated detection of semantic exceptions, and the partially automated derivation of the implications for running workflow instances. The approach is mainly motivated by our experiences with the system HEMATOWORK supporting long-term treatment in cancer therapy. One basic problem of domains such as oncology is the enormous amount of diagnostic and monitoring data, and that a significant number of patients suffer from very specific drug-side effects and other exceptions, so that their treatment workflows have to be modified partially. Therefore, the approach basically intends first to filter exception events out from "normal" events by a rule base, second to automatically derive which workflow instances are affected w.r.t. which control flow areas, and third to automatically adjust – as much as possible – the affected areas w.r.t. their control flow (by dropping and adding new activities). For this purpose, two basic control flow modification algorithms (*drcd*-and *p*-algorithm) have been introduced. Even if they fail (if, for example, the *p*-algorithm

[12] Please recall that the cardinality of N_E is > 1 iff the interruption occurred during a *parallel* path execution. Therefore, A_{new} must be integrated only into *one* sequence.

is not able to determine an insertion point meeting all constraints), and modification control must be shifted to the domain expert, the system can provide at least information in which region the insertion should take place.
Of course, argument could be made, that the cascade of deletion and insertion operations may lead to incorrect workflows. The counter-argument is, that as long as the original workflow definitions *and* the underlying rule base are consistent, the modified workflow should be consistent as well. Consistency of rule bases has been addressed by several artificial intelligence approaches (e.g. [SSS82]), while semantic consistency of (human-defined) workflows still is an open research problem, which is, however, orthogonal to our approach of translating rules as procedural fragments into workflow modification operations.

Furthermore, the modification agents described need a lot of declarative knowledge to derive modification implications when events occur. While this may not be readily available in all domains, this approach is well suited for medical domains where knowledge bases are commonly available and can thus be extended for the additional purposes. A further characteristic of our approach is the high percentage of temporal aspects, as we, for example, have to determine appropriate modification and interval-covering workflow regions. Currently, we use user-defined "temporal" objects to express heuristic temporal constraints, but a more explicit system-supported approach by using temporal databases in connection with temporal logic could, from the formal point of view, model some aspects better.

At the moment, we are only able to delete or insert single nodes according to derived predicates which definitely could have been evaluated to *true*. However, as changing a workflow instance must be viewed as a very time-consuming process, it is desirable to support larger workflow changes. For this, it can be useful to insert larger subgraphs containing activity nodes which may not necessarily will be processed but are likely – because of the particular type of E – to become relevant during further execution of the workflow instance i_W. For example, if it has been derived that a drug *D* should be applied because of an event, and if it is known that most of the patients receiving this drug have some side-effects after a few days implying additional measures, it makes sense to directly integrate this additional activities into the control flow, as it is likely that the patient will need at least one of these additional activities. Further work will concentrate on these topics.

Furthermore, the implementation of HEMATOWORK is still one of the major efforts to be addressed. Currently, we have developed the patient database *DB* based on ORACLE and a knowledge base *KB* realized with O_2. Rules and rule processing modules (*RI*) are implemented with the rule shell CLIPS (C-LANGUAGE INTEGRATED PRODUCTION SYSTEMS; [GR93])[13]. The implementation of the engine still is an open problem, as the commercial workflow systems we inspected do not provide sufficient support w.r.t. run-time interrupts and dynamic changes. Therefore, at the moment, an own implementation currently is addressed, based on the CORBA implementation of IONA ORBIX.

[13] Although there exists an available prototypical implementation of F-Logic (FLORID [FHK+97]), we had to implement our rule-related modules with a low-level rule shell (i.e. CLIPS) because of the restricted API capabilities of FLORID.

Acknowledgements

We are grateful to M. Reichert, who has given valuable comments especially w.r.t. the graph modification algorithms. Furthermore, we want to thank the anonymous reviewers for their comprehensive suggestions to improve the paper.

References

ASE+96 Attie, P.; Singh, M.P.; Emerson, E.A.; Sheth, A.; Rusinkiewicz, M.: *Scheduling Workflows by Enforcing Intertask Dependencies*. Distributed Systems Engineering Journal, vol 3, 1996: 222-238.

BJ96 Bussler, C.; Jablonski, S.: *Die Architektur des modularen Workflow-Management-Systems MOBILE*. In [Vos96]: 369-388.

BK94 Bonner, A.J.; Kifer, M.: *An Overview of Transaction Logic*. Theoretical Computer Science, 133: 205-265.

BK95 Bonner, A.J.; Kifer, M.: *Transaction Logic Programming.* Technical Report CSRI-323, Computer Science Research Institute, University of Toronto, 1995.

CGP+97 Casati, F.; Grefen, P.; Pernici, B.; Pozzi, G.; Sanchez, G.: *WIDE Workflow Model and Architecture*. Technical Report, University of Milano, 1997.

DKÖ+98 Dogac, D.; Kalinichenko, L.; Özsu, T.; Sheth; A. (eds.): *Workflow Management Systems and Interoperability*. Springer, Berlin, 1998.

DKR+98 Davulcu, H.; Kifer, M.; Ramakrishnan, C.R.; Ramakrishnan, I.V.: *Logic-Based Modeling and Analysis of Workflows.* Proc. PODS98: 25-33.

FHK+97 Frohn, J.; Himmeröder, R.; Kandzia, P.-Th.; Lausen, G.; Schlepphorst, C.: *FLORID - Ein Prototyp für F-Logik*. Proc. BTW97, Ulm, 1997: 100-117.

GR93 Giarratano, J.; Riley, G.: *Expert Systems: Principles and Programming*. PWS Publishing Company, 1993.

HSS96 Heinl, P.; Schuster, H.; Stein, K.: *Behandlung von Ad-hoc-Workflows im MOBILE Workflow-Modell.* ITG Fachbericht, STAK '96 – Softwaretechnik in Automation und Kommunikation, München, März, 1996.

KLW95 Kifer, M.; Lausen, G.; Wu, J.: *Logical Foundations of Object-Oriented and Frame-Based Languages*. Journal of the ACM, 42(4), 1995: 741-843.

MH98 Müller, R.; Heller, B.: *A Petri Net-based Model for Knowledge-based Workflows in Distributed Cancer Therapy*. Proc. Intern. EDBT98 Workshop on Workflow Management Systems, Valencia, Spain, March 1998.

MHL+98 Müller, R.; Heller, B.; Löffler, M.; Rahm, E.; Winter, A.: *HematoWork: A Knowledge-based Workflow System for Distributed Cancer Therapy*. Proc. of the GMDS98, Bremen, Sep. 98.

Mül99 Müller, R.: *Rule-based Interruption and Modification of Workflow Instances. Technical Report*. Department of Computer Science, Leipzig

University 1999.

MSN+97 Müller, R.; Sergl, M.; Nauerth U. et al.: *TheMPO: A Knowledge-Based System for Therapy Planning in Pediatric Oncology*. Computers in Biology and Medicine vol. 27(3), 1997:177-200.

MTD+96 Musen, M.A.; Tu, S.W.; Das, A.K et al.: *EON: A Component-Based Architecture for Automation of Protocol-Directed Therapy*. Journal of the American Medical Informatics Association 1996 (3): 367-388.

RD98 Reichert, M.; Dadam, P.: *ADEPT$_{FLEX}$ - Supporting Dynamic Changes of Workflows Without Losing Control*. Journal of Intelligent Information Systems 10, 1998: 93-129.

She97 Sheth, A.: *From Contemporary Workflow Process Automation to Adaptive and Dynamic Work Activity Coordination and Collaboration*. Proc. DEXA Workshop on Workflow 1997.

SK98 Sheth, A.; Kochut, K.: *Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems*. In [DKÖ+98]: 35-60:

SSS82 Suwa, M.; Scott, A.C.; Shortliffe, E.H.: *An approach to verifying completeness and consistency in a rule-based expert system*. AI Magazine 3, 1982:16-21.

Vos96 Vossen, G.: *Geschäftsprozeßmodellierung und Workflow-Management: Modelle, Methoden, Werkzeuge*. Thomson, Bonn, 1996.

VW98 Vossen, G.; Weske, M.: *The WASA Approach to Workflow Management for Scientific Applications*. In [DKÖ+98]: 145-164.

Wes98 Weske, M.: *Flexible Modeling and Execution of Workflow Activities*. Proc. 31st Hawaii Int'l Conf. on System Sciences (HICSS–31), Software Technology Track (Vol VII), 1998: 713–722.