Enabling YAWL to Handle Dynamic Operating Room Management

Sebastian Schick, Holger Meyer, Markus Bandt, and Andreas Heuer

Database Research Group
University of Rostock
Germany
{schick,hme,mb,heuer}@informatik.uni-rostock.de

Abstract. Clinical workflows are known to be often complex and have to be handled very flexible due to the patients individual anamnesis and state of health. Certain situations require urgent changes of the previously planned process at run time. Some choices to be made in this context depend very much on the data from clinical backend systems. Thus, data and processes cannot be treated independently of each other. We present an approach for flexible, data centric workflows. It extends the control-flow perspective of a workflow management system with new concepts for handling process adaption at run-time. The approach combines the method of late modeling with declarative concepts and underspecification. Due to constraints on data from clinical backend systems, process adjustment is triggered at certain points of the process and is then performed at runtime.

Key words: Workflow, Flexibility, Healthcare, Perioperative Process, Yawl, Flexible Workflow Modeling

1 Introduction

In medical and especially in clinical enviroments the demands not only on increased quality of service but also on better cost efficiency for treatment and care grows constantly. That's why resident doctors and hospitals are obliged to optimize the patient treatment cycle in any possible way. In general a process aware workflow perspective provides opportunities to improve quality of service as well as cost efficiency and thus is progressively acknowledged and embraced by the medical community.

The modelling of patient treatment processes is quite challenging though. Work in this domain is known to be complex and highly flexible. Independant ways to work combined with the different skill levels of the staff are hard to quantify and therefore related workflows have to reflect the differences between these approaches at model level. Beyond that, the patients distinct anamnesis, state of health and aetiopathology are decisive for the course of action. These aspects generally are recorded as structured data which can be properly interpreted in corresponding processes at instance level.

A process oriented data model and an adequate communication protocol within medical environments is the HL7 (Health Level Seven) and CDA (Clinical document architecture) standards. For example HL7 defines structured messages for each high level event and each major task connected with patient treatment and provides a basic structure for clinical documents as well. HL7 compliant integration of data in clinical workflows is appropriate for proactive data provision and thus can enhance medical decision support.

Using an example from the perioperative process we introduce an approach to dynamically adapt the control flow of a process at instance level with respect to (HL7) data from clinical back-end systems. According to the taxonomy from [1] this technique can be classified as *late modeling* combined with declarative elements and under-specification.

The paper is organized as follows. In Sect. 2 we introduce the perioperative process and provide a motivating example together with requirements for flexibility in this domain. Then we discuss related work. In Sect. 3, we present a method for dynamic specification of the perioperative process as well as for the related composition of flexible process parts. Section 4 describes the transformation of our intermediate format into the YAWL process language. In Sec. 5 we illustrate our approach by an example.

2 Flexibility in the Perioperative Process

2.1 The Perikles Project

In the context the PERIKLES[1] project, we analyzed perioperative processes concerning the demands of flexibility and the data flow ([2, 3]). As part of the results of PERIKLES the YAWL engine got extended in several ways. A resource data model has been developed and a corresponding planning service has been implemented as well as a scheduling service for these resources ([4]) and a framework for improved, transactional access to external data sources.

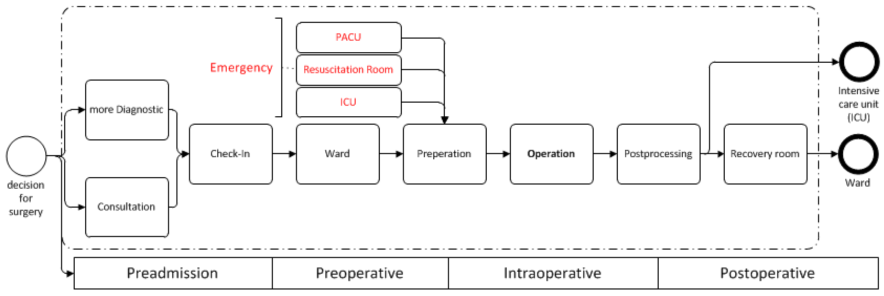

Fig. 1. Generalized perioperative process (adapted from [5])

[1] The PERIKLES project (`http://www.perikles.org/`) is partially funded by the German Federal Ministry of Education and Research under contract 01IS09009B.

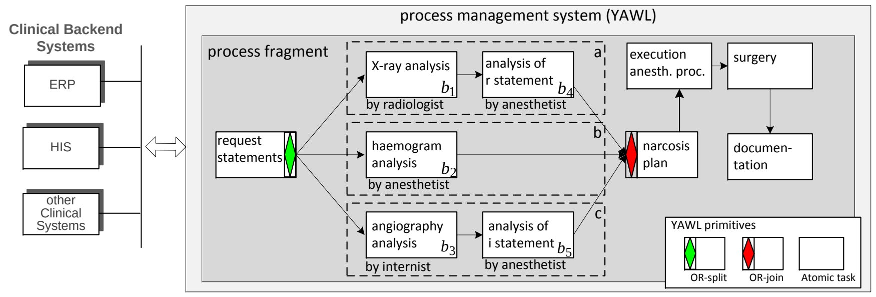

Fig. 2. Example fragment of the anaesthesia workflow

In PERIKLES we extended this scope by the preadmissional timespan like shown in Fig. 1. However, in this paper we concentrate on the processes on the day right before the surgical treatment of a patient and the day of the surgery itself. This includes the preoperative preparation notably the examination of the patient by an anaesthetist, the preparation of the patient at the preoperative day and the medication, the transfer to OR area and the anaesthetic preparation and treatment at the day of the surgery. For simplicity we consider the surgical treatment as an atomic task. The postoperative period includes the completion of the narcosis and immediate postoperative care at the Post Anaesthesia Care Unit (PACU) or at the Intensive Care Unit (ICU). The process shown in Fig. 1 illustrates a patient centered perspective. There is also the need of documentation which includes all diagnostic data, every planned (prescribed) action and every executed action in the perioperative process.

2.2 Requirements on Flexibility and Data Access

Among other things the results of the requirements analysis in the context of PERIKLES showed that several recurring classes of flexibility structures can be found in these processes.

These are namely *partial order* (some tasks have to be executed in a specified order while other tasks can be executed before or after any given task in the sequence – **Requirement 1**), *optional tasks* (**Requirement 2**), *repetitive execution of complex sub-processes* (**Requirement 3**) and *alternative tasks* (**Requirement 4**). In [6] we described the corresponding processes as well as the identified structures more detailed and presented an implementation approach using the workflow management system (WFMS) YAWL. Due to space limitations in this paper we will provide just one example which includes several of the mentioned structures.

In Fig. 2 a fragment of the perioperative process is shown as it was implemented according to guidelines we provided in [6]. The process fragment in the gray box is implemented in YAWL [7]. The clinical backend systems provide diagnostic findings which are accessible e.g. via HL7 compliant interfaces and can be integrated into the workflow net using the data access extension mentioned in Sec. 2.1. In general the access on external data sources is required to be independent from underlying systems (**Requirement 5**) which is ensured by the extension. In our example though the backend systems are integrated by using the HL7 standard.

Depending on the individual state of health of a patient several diagnostic results are needed to be considered at the planning of the anaesthesia. X-ray pictures (fragment b_1) of the chest and haemograms (fragment b_2) are mandatory while the angiography shown in the picture is an example for an optional diagnostic examination result (*Req. 2*). In Fig. 2 the users are in control of the temporal order in which the three diagnostic results are analyzed. But in case of X-ray and angiography (fragment b_3) the specialists have to make a statement about the results first before the results are enabled to the anaesthetist so there is partial order of execution necessary (*Req. 1*).

After the diagnostic results and statements are analyzed (fragments b_4,b_5) the anaesthesia can be planned. On the day of surgery the planned anaesthesia is usually put into effect. After the surgical procedure the whole process has to be documented by the participating users.

This example shows one possible implementation of the workflow which is quickly build, stable and especially easy to maintain as long as the number of parallel paths is low. Though there may be a complete blood count (not shown in the picture) needed instead of a haemogram (which is a subset of tests included in the complete blood count) so there may be alternative paths of execution involved (*Req. 4*). Nevertheless, the corresponding task is meant to handle both diagnostic results since they are of the same type. Furthermore, it could be necessary to check all daily blood count results from the patient over the past week (*Req. 3*) which is in Fig. 2 represented by just one task. So this is a rather pragmatic approach which comes with the trade-off that the implementation is not quite as exact and as flexible as the real process in the hospital is.

2.3 State of the Art

Several work has been done in the area of supporting healthcare processes using workflow management systems. Of these, few especially were concerned with the perioperative process. Related work can be found in the general area of flexible business process management systems [8, 9, 10, 11]. Few papers explore flexibility in workflows for healthcare, e.g. [12, 13, 14].

Reijers et al. [12] identifies several flexibility patterns but concentrates on the outpatient management in a Dutch hospital. Furthermore, how current workflow system would support such patterns is also part of the analysis.

Müller, Greiner, and Rahm [13] present a system called *AgentWork* providing support for automated workflow adaption. To cope with exceptions during workflow execution an ECA rule approach based on temporal logic was introduced. The event monitoring is described using ActiveTFL (Active Temporal Frame Logic) which is mapped to database triggers. *AgentWork* is highly related to the underling process management system ADEPT [10] which offer a rich set of change operations supporting dynamic structural adaptations ([1]). However, the trigger mechanism allows only monitoring state transitions. But we need at certain times the exact state of data sources. Additionally, the change of process instances according to the principle of ADEPT is very expensive. Frequent changes in the process model, which may need to be verified by the users, is not acceptable for our application.

Hallerbach et al. [15] configure process models extending the process modeling language. Configuration elements within the modeling language are used to configure the process model. The Provop approach supports flexibility during execution by switching between different process variants. As this method is very costly, our approach compose the required model at runtime.

Pockets of flexibility [11] uses the concept of *open instances*. Within the process model pockets of flexibility were defined within a core process. A pocket is a special build activity which composes activities depending on different constructs (e.g. fork sequence, etc.). Just as our approach it is according to [1] assigned to the late modeling concept. However, the composition is left to the user and is restricted afterwards by conditions. Also conditions related to external environment are not considered.

Flexibility as a service is offered by the WFMS YAWL [16, 8, 9]. The Worklet approach [8] offers a set of self-contained sub-processes. Selection rules (Ripple Down Rules) are used to pick up a Worklet. However, dynamics are restricted to flexible selection of ready-made sub-processes which corresponds to the concept of late binding introduced in [1]. DECLARE [9] avoids the disadvantages of Worklets by using declarative models describing loosely-structured processes. The approach also has drawbacks with data integration. Constraints are only defined between tasks and task parameters. In addition, process models are very complex, if many rules have to be used to describe the execution in detail.

We present a new approach to support flexible workflows in the clinical environment. Therefore, our flexible workflows will be adapted in dependency of the current state of data generated by various clinical systems.

3 Flexible Data Aware Workflows

3.1 Dynamic Process Specification

Within the PERIKLES project, HL7 messages, generated by various clinical systems, will be persisted as XML type documents in a XML Data Base. Our processes will be adapted in dependency of the current state of theses HL7 messages and other XML type data sources during runtime using a *dynamic dispatch* of activities. Therefore, our approach observes messages and data which are broadcasted via different channels and specifies where corresponding changes in the process should take effect, i.e. where dynamically generated sub-process are executed. We use a notation of a core process which is extended with special *observer* and *generator* tasks. In order to achieve flexibility within the perioperative process the full specification of the process model is completed at runtime. To illustrate our approach Fig. 3 outlines the newly introduced concepts.

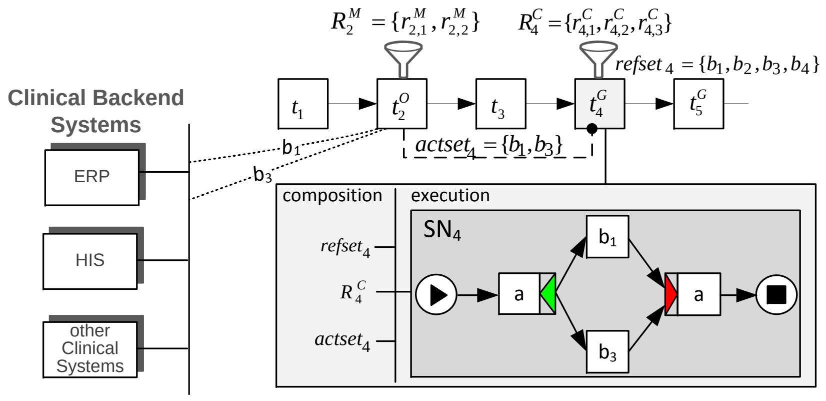

Fig. 3. Dynamic process specification

Bricklets b_i are the building blocks for re-using specific activities which are bundled into sub-processes. They are not directly part of the process specification but will be executed at well defined points. The bricklets are a mean for separating the application specific process from data specific parts which rely on up-to-date data.

Definition 1 (Bricklet). *Let* $B = \{b_1, b_2, \cdots, b_i\}$ *with* $i \in N$ *be the set of* bricklets. *A bricklet* b_i *is a valid process model, which contains at least one task definition. Bricklets will be assembled into subnets* SN_i.

Observer tasks t_i^O define points within the workflow where the actual broadcasted messages and data will be investigated. Usually, observer tasks are inserted after activities which expected to cause major changes of the data. The task t_2^O in Fig. 3 denotes an observer task which makes use of the *matching rule set* (R_2^M). The rules ($r_{2,1}^M, r_{2,2}^M$) specify which activities should be added or removed from the process if there are certain parts in the HL7 message or if they are absent.

Whether a HL7 message fragment exists or not is determined using the path expression $pexpr_j$ and the function $match(pexpr_j)$. If the expression gets evaluated true corresponding bricklets are added to or removed from the construction set $actset_4$ of *generator tasks* t_4^G. Furthermore, the selected fragments are returned as $pfrag_j$.

Definition 2 (Observer Task). *Let* T^O *denote the set of* observer tasks $t_i^O = (R_i^M, match)$ *with* $i \in N$. *Then:*

- $R_i^M = \{r_{i,1}^M, r_{i,2}^M, \cdots, r_{i,j}^M\}$ *with* $j \in N$ *is a set of matching rules.*
- $r_{i,j}^M : match(pexpr_j) \mapsto (op, t_m^G, B_j, pfrag_j)$ *is a matching rule.*
- $pexpr_j$ *is used to specify data parts expected within the data. It is basically a XPath expression.*
- $op = \{add, delete, merge, undo\}$ *is the set of change operations.*

– $t_m^G \in T^G$ *is a generator task.*
– $B_j \subseteq refset_m$ *is a set of predefined bricklets.*
– $pfrag_j$ *contains the resulting XPath 1.0 nodeset using* $pexpr_j$*.*

Matching rules R_i^M associate parts of a message (based on content and/or structure) with a set of bricklets (activities) and define points in the control flow where the activities should be scheduled. So, XPath expression will describe the parts within message instances which should (not) match and trigger activities in the subsequent workflow. Sometimes the existence or absence of a message will not only add but remove also scheduled activities depending whatever the default behavior may be, e.g. if the patient withdraw the prior informed consent (PIC) several treatment activities will most likely be cancelled immediately.

Each $pexpr_j$ within a matching rule $r_{i,j}$ is closely associated with an operation *op*. Where an operation *op* may only use bricklets from set $refset_i$.

Definition 3 (Change Operation). *Let* $op = \{add, delete, merge, undo\}$ *denotes the set of possible operations to manipulate the construction set* $actset_i$ *(hereinafter actset) of generator tasks* t_i^G*. Then:*

– *The* add *operation appends the set of activated bricklets* B_j *to the activation set actset.b. For each activated bricklet also the corresponding number* $count(pfrag_j)$ *is stored in* $actset.f(b_k)$*.*
$add(t_m^G, B_j, pfrag_j) \mapsto \forall\ b_k \in B_j : \{actset.b = actset.b \cup b_k$
$\wedge\ actset.f(b_k) = count(pfrag_j)\}$
– *The* merge *operation appends a set of predefined bricklets* B_j *only into actset.b if they are not member of it. Virtual, this operation updates* $actset.f(b_k)$
$merge(t_m^G, B_j, pfrag_j) \mapsto \forall\ b_k \in (B_j \cap actset.b) : actset.f(b_k) = count(pfrag_j)$
– *The* delete *operation removes bricklets from actset.*
$delete(t_m^G, B_j, pfrag_j) \forall b_k \in B_j : \{(actset.b = actset.b \setminus \{b_k\})$
$\wedge delete(actset.f(b_k))\}$
– *Let* B_j^{-1} *be the compensation of* B_j *then the* undo *operation appends a set of compensating bricklets* B_j^{-1} *to the activation set actset to rollback operations* B_j *after a data fragment was removed from the document.*
$undo(t_m^G, B_j) \mapsto \forall\ b_k \in B_j : \{actset.b = actset.b \cup b_k\}$

Generator tasks[2] t_i^G specify points within the flow of control where bricklets are combined at run-time to build up a subnet of activities, e.g. SN_4 in Fig. 3. The resulting subnet is then deployed and executed. Essentially, the generator tasks are responsible for dynamic dispatching the activities/bricklets like selecting and executing method calls in object-oriented systems. For building up the subnets a set of composition rules (R_i^C) and set of scheduled activities ($actset_4$) is used. The scheduled activities must belong to a set of allowed activities ($refset_4$) per distinct generator task. If a generator task is executed within the process, it has to compose a valid execution order for the activated bricklets. The generator tasks is the anchor point for providing flexibility at the process instances level.

[2] It resembles the idea of *pockets of flexibility* introduced in [11].

Definition 4 (Generator Task). *Let T^G be the set of* generator tasks $t_i^G : (R_i^C, actset_i, refset_i) \mapsto SN_i$ *with* $i \in N$. *Then:*

- SN_i *is a valid subnet executed if* t_i^G *is processed within the control flow.*
- $R_i^C = \{r_{i,1}^C, r_{i,2}^C, \cdots, r_{i,j}^C\}$ *with* $j \in N$ *defines a set of construction rules which are used for the generation of a valid* SN_i.
- $actset_i$ *is a set of active bricklets* (b_k) *and corresponding number of data fragments* ($f(b_k)$) *chosen by different* observer tasks t_m^O.
- $refset_i$ *defines all bricklets allowed for* t_i^G.

Construction rules R_i^C define relationships between bricklets and how they are combined into a resulting control flow. If a bricklet is a pre-requisite for another, a sub-sequent order can be specified. Further, a bricklet can be executed sequential or parallel n-times. If not stated otherwise, bricklets can be executed arbitrarily and in parallel. The construction rules are used to generate a valid subnet SN_i during runtime, which have to be instantiated for t_i^G at runtime.

Definition 5 (Construction Rules). *Let* R_i^C *be the set of construction rules* $r_{i,j}^C$. *A construction rule* $r_{i,j}^C \in \{b_k \prec b_l, b_k \overset{n}{\prec}, b_k \overset{n}{||}\}$ *defines how a* bricklet b_k *is inserted into the subnet* SN_i *iff* $b_k \in actset_i$.

- $b_k \prec b_l$: *Iff bricklet* $b_l \in actset_i$, b_l *is immediately executed after* b_k.
- $b_k \overset{n}{\prec}$: *The bricklet* b_k *will be inserted sequentially* n *times.*
- $b_k \overset{n}{||}$: *The bricklet* b_k *will be inserted* n *times in parallel.*

By using these concepts, we avoid complex process structures. The primary process specification is a model of the application's point of view. Wherever message specific activities have to be carried out, they are hidden by generator tasks and descriptive matching and composition rules. These rules determine the dynamic execution of a re-usable set of message specific activities.

3.2 Composing Sub-processes

After the activation of bricklets, which is done by the observer tasks, the construction of a valid sub-process has to be controlled by the generator task. Therefore, we provide an algorithm for combining bricklets b_k into a valid subnet using the construction rule set R_i^C. The composition is done during runtime to offer a flexible generation of subnets.

Since $actset_i$ changes during runtime, SN_i has to be generated only when t_i^G is activated. This is a two-step procedure. First a directed acyclic graph is created with all activated bricklets $b_n \in actset_i$. In the second step we transform the graph into a valid sub-process (YAWL subnet).

Definition 6. *Let* $G_i^C = (V, E)$ *be a digraph.* V *is the set of vertices and* E *is the set of directed edges. The graph* G_i^C *contains only one starting vertex "start"* $\in V$ *and one ending vertex "end"* $\in V$.

Definition 7. *The indegree $deg^-(b_k)$ is the number of head endpoints for bricklet b_k. The outdegree $deg^+(b_k)$ is the number of tail endpoints for b_k.*

Listing 1 shows a simple algorithm to calculate SN_i. Since the subnet is composed during runtime no deferred choice is needed and no composition rule is mapped to OR-splits, too. We avoid cycles in the constructed subnet graph by enforcing acyclicity of the construction rule set R_i^C at modeling time[3].

Listing 1. Algorithm to calculate subnets

```
initialize $G$ with $G.V = \{start, end\} \cup actset_i$ and $G.E = \{\}$
foreach $r_{i,j}^C \in R_i^C$ {
  if $r_{i,j}^C$ equals $b_k \prec b_l$ and $\{b_k, b_l\} \subseteq actset_i$ {
    add directed edge $(b_k, b_l)$ }}
foreach $r_{i,j}^C \in R_i^C$ {
  //expand replaces $b_k$ by $n$ nodes $b_{k.m}$ of type $b_k \wedge m \in \{1, \ldots, n\}$
  if $r_{i,j}^C$ equals $b_k \overset{n}{\prec}$ and $b_k \in actset_i$ {
    $n$ = $actset_i.f(b_k)$; expand($b_k, n$);
    add directed edge between successive $b_{k.m}$ }
  if $r_{i,j}^C$ equals $b_k \overset{n}{||}$ and $b_k \in actset_i$ {
    $n$ = $actset_i.f(b_k)$; expand($b_k, n$)
    if $\exists r_{i,m}^C \in R_i^C \wedge r_{i,m}$ equals $b_k \prec b_l$ {
      foreach $b_{k.o} \in \{b_{k.1}, \ldots, b_{k.n}\}$ { add directed edge $(b_{k.o}, b_l)$ }}
    if $\exists r_{i,m}^C \in R_i^C \wedge r_{i,m}$ equals $b_l \prec b_k$ {
      foreach $b_{k.o} \in \{b_{k.1}, \ldots, b_{k.n}\}$ { add directed edge $(b_l, b_{k.o})$ }}}}
foreach $b_k \in G.V$ {
  if $deg^-(b_k) = 0$ { add directed edge $(start, b_k)$ }
  if $deg^+(b_k) = 0$ { add directed edge $(b_k, end)$ }}
```

4 Implementation using Yawl and Component Services

The approach presented above was exemplified using YAWL [17] and the corresponding WFMS YAWL. Two YAWL *Custom Component Services* for the observer and generator task types were implemented. We have extended the YAWL editor to describe the matching and construction rule sets. This allows for modeling everything within the standard YAWL environment. The bricklets are implemented as YAWL nets which contain always a start and end condition.

After constructing a digraph for subnet SN_i within generator task t_i^G, the graph is tansformed into a valid YAWL net. This YAWL net then gets executed by the WFMS. The transformation is based on rules $R_{1\ldots6}$ shown in Fig. 4. In the resulting net each bricklet is represented by a composite task. This tasks in turn is a container for the bricklet process.

R_1 maps the *start* node to a YAWL *Input Condition* where the process starts.
R_2 maps the *end* note to a *Output Condition* where the process ends.

[3] This can be done by applying R_i^C on $refset_i$

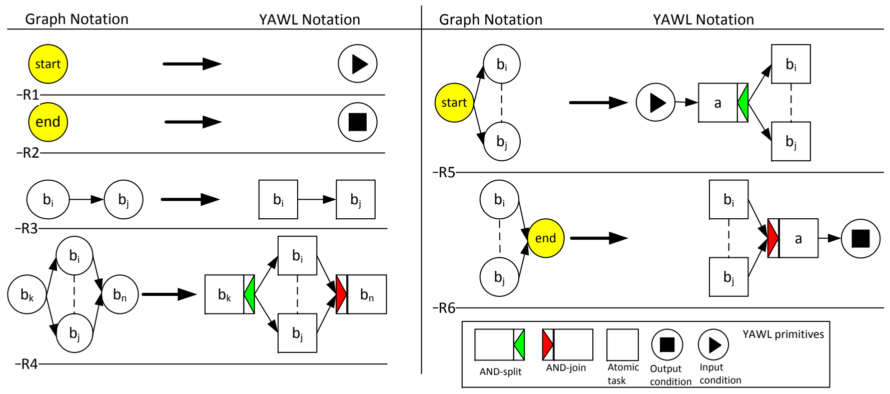

Fig. 4. Graph to YAWL mapping rules

R_3 maps a sequential path from node b_i and b_j to corresponding tasks b_i, b_j.

R_4 maps the split of node b_k to a *And-Split task* b_k. Nodes b_i, b_j are mapped to corresponding tasks b_i, b_j. Node b_n is mapped to a *And-Join task* b_n.

R_5 is for circumstances where the start node *start* is part of a parallel execution. *start* is mapped to a *Input Condition* together with a *And-Split task* a. a is a dummy task. Nodes b_i, b_j are mapped to corresponding tasks b_i, b_j.

R_6 is for the same situation as R_5, if the end node *end* is part of a parallel execution. *end* is mapped to a *Output Condition* together with a dummy *And-Join task* a. Nodes b_i, b_j are mapped to corresponding tasks b_i, b_j.

5 Implementing the Sample Scenario

In Fig. 5 (a) the process from Fig. 2 is realized using our approach. Task *request statement* (t_1) is modeled as an observer task which controls the dynamic dispatch of activities in generator task *planning* t_2. Task t_2 provides the set of selectable bricklets $refset_2 = \{b_1, b_2, b_3, b_4, b_5\}$, which are correspond to the tasks in Fig. 2. The observer task t_1 uses a set of matching rules $R_1^M = \{r_{1,1}^M, r_{1,2}^M, r_{1,3}^M\}$ with $r_{1,1}^M = add(t_2, b_2,$ "`//OBR[../PID/PID.3/CX.1='123'][OBR.4/CWE.1='X-ray']`") and $r_{1,2}^M = add(t_2, \{b_1, b_4\},$ "`//OBR[../PID/PID.3/CX.1='123'][OBR.4/CWE.1='haemogram']`")[4]. For the sub-process construction in t_2, the rule set $R_1^C = \{b_1 \prec b_4, b_1 \overset{2}{\prec}, b_2 \overset{2}{||}\}$ is used. They reflect the order of the activities in Fig. 2. Due to matching rules $r_{1,1}^M$, $r_{1,2}^M$ the observer task t_1 activates the bricklets in $actset_2 = \{b_1, b_2, b_4\}$, which is a subset of $refset_2 = \{b_1, b_2, b_3, b_4, b_5\}$. Fig. 5 (b)–(d) depict the construction of digraph G. First, all activated bricklets (b_1, b_2, b_4) will be inserted into $G.V$ (Fig. 5 (b)).

[4] The XPath queries on the HL7 messages match for the patient 123 and if they contain a X-ray or a haemogram.

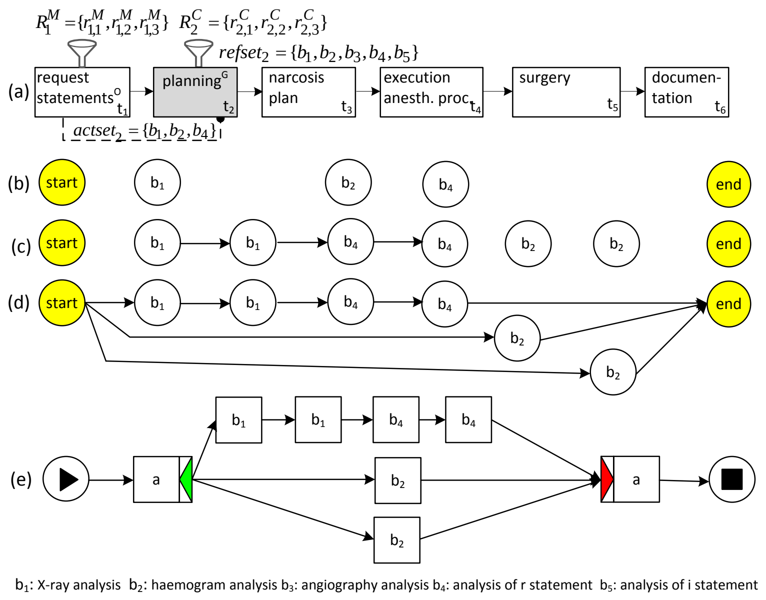
$R_1^M = \{r_{1,1}^M, r_{1,2}^M, r_{1,3}^M\}$
$R_2^C = \{r_{2,1}^C, r_{2,2}^C, r_{2,3}^C\}$
$refset_2 = \{b_1, b_2, b_3, b_4, b_5\}$
(a)
request statements^O t_1
planning^G t_2
narcosis plan t_3
execution anesth. proc. t_4
surgery t_5
documen-tation t_6
$actset_2 = \{b_1, b_2, b_4\}$
(b)
(c)
(d)
(e)
start
end
b_1
b_2
b_4
a
b_1: X-ray analysis b_2: haemogram analysis b_3: angiography analysis b_4: analysis of r statement b_5: analysis of i statement

Fig. 5. Digraph construction and transformation into a Yawl subnet

The application of rules $b_1 \prec b_4$ (edge from b_1 to b_4), $b_1 \overset{2}{\prec}$ (edge from first b_1 to second b_1) and $b_2 \overset{2}{||}$ (b_2 is duplicated) is shown in (c). We assume that $count(pfrag_j)$ returns always 2. The final step (d) connects all vertices with $deg^-(b_i) = 0$ or $deg^+(b_i) = 0$ with the "start" and "end" nodes. The Yawl subnet (e) results from transforming digraph G into Yawl. This subnet gets deployed and executed at run-time by generator task t_2.

6 Conclusion and Future Work

Process support in the perioperative process is a field that bears much practical relevance. We provided an approach for a flexible perioperative process. With respect to other approaches, we achieve flexibility by monitoring data changes and specifying where corresponding changes should take effect. This is done by extending the Wfms Yawl with observer tasks which monitor these changes. Generator tasks allow for flexible execution of process instances. Our technique presented here can be described best as a late modeling, descriptive approach using under-specification [1].

The added value of our approach is that both context conditions on external data and the resulting changes of the process instances are described within one process model. Additionally, all components of our approach will be provided and executed within an extension of the Wfms Yawl. A prototype that implements the approach will finally allow a detailed case study and evaluation.

In future research we will extent relationships between data operations and rule sets for process constructions to provide more freedom in combining bricklets (process fragments) into sub-processes. One of the challenges concerns the automatic generation of construction rules. Also data dependencies between bricklets have to be considered in more detail.

References

1. Weber, B., Sadiq, S., Reichert, M.: Beyond rigidity — dynamic process lifecycle support. Computer Science — Research and Development **23** (2009) 47–65
2. Kühn, R., Bandt, M., Dittmar, A., Meyer, H., Forbrig, P.: Hops: modeling flexible, clinical processes as the basis of workflow-based assistance system (german). In: USEWARE 2010. Number 2099 in VDI-Berichte/VDI-Tagungsbände (2010) 77–86
3. Kühn, R., Dittmar, A., Forbrig, P.: Alternative representations of workflow control-flow patterns using hops. In: LNBIP. Volume 64. Springer (2010) 115–129
4. Ouyang, C., Wynn, M.T., Fidge, C., ter Hofstede, A.H.M., Kuhr, J.C., Becker, T.: Workflow support for scheduling in surgical care processes. In: accepted paper at ECIS 2011. (2011)
5. Sandberg, W.S., Ganous, T.J., Steiner, C.: Setting a Research Agenda for Perioperative Systems Design. Surgical Innovation **10**(2) (2003) 57–70
6. Bandt, M., Kühn, R., Schick, S., Meyer, H.: Beyond flexibility - workflows in the perioperative sector of the healthcare domain. ECEASST **37** (2011)
7. van der Aalst, W.M.P., ter Hofstede, A.: Yawl: Yet another workflow language. Information Systems **30**(4) (2005) 245–275
8. Adams, M., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Worklets: A service-oriented implementation of dynamic flexibility in workflows. In: OTM Conferences (1). (2006) 291–308
9. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: Declare: Full support for loosely-structured processes. In: EDOC. (2007) 287–300
10. Reichert, M., Dadam, P.: ADEPT$_{\mathsf{flex}}$ — Supporting Dynamic Changes of Workflows Without Losing Control. J. Intell. Inf. Syst. **10**(2) (1998) 93–129
11. Sadiq, S.W., Orlowska, M.E., Sadiq, W.: Specification and validation of process constraints for flexible workflows. Inf. Syst. **30**(5) (2005) 349–378
12. Reijers, H.A., Russell, N.C., Van Der Geer, S.b., Krekels, G.A.M.c.: Workflow for healthcare: A methodology for realizing flexible medical treatment processes. Lecture Notes in Business Information Processing **43 LNBIP** (2010) 593–604
13. Müller, R., Greiner, U., Rahm, E.: Agentwork: A workflow system supporting rule-based workflow adaptation. Data and Knowledge Engineering **51**(2) (2004) 223–256
14. Mans, R.S., Russell, N.C., van der Aalst, W.M.P., Bakker, P.J.M., Moleman, A.J., Jaspers, M.W.M.: Proclets in healthcare. Journal of Biomedical Informatics **43**(4) (2010) 632–649
15. Hallerbach, A., Bauer, T., Reichert, M.: Capturing variability in business process models: the provop approach. Journal of Software Maintenance **22**(6-7) (2010) 519–546
16. van der Aalst, W.M.P., Adams, M., ter Hofstede, A.H.M., Pesic, M., Schonenberg, H.: Flexibility as a service. In: DASFAA Workshops. (2009) 319–333
17. van der Aalst, W.M.P., ter Hofstede, A.H.M.: Yawl: yet another workflow language. Information Systems **30**(4) (2005) 245–275