Process Execution and Enactment in Medical Environments

Bernard Lambeau, Christophe Damas and Axel van Lamsweerde

ICTEAM Research Institute, Université catholique de Louvain
{bernard.lambeau, christophe.damas, axel.vanlamsweerde}@uclouvain.be

Abstract. Process models are increasingly recognized as an important asset for higher-quality healthcare. They may be used for analyzing, documenting, and explaining complex medical processes to the stakeholders involved in the process. Models may also be used for driving single processes or for orchestrating multiple ones. Model-driven software technologies therefore appear promising. In particular, process enactment provides software-based support for executing operational processes. A wide variety of possible enactment schemes are available in medical environments, e.g., to maintain daily medical worklists, to issue warnings or reminders in specific process states, to schedule tasks competing for resources, to provide on-the-fly advice in case of staff unavailability, and so forth. Such variety of possible process enactments calls for a common conceptual framework for defining, comparing, classifying, and integrating them. The paper introduces such a framework and describes a number of patterns for process execution and enactment based on it. These patterns result from a simple generic, goal-oriented model of medical process execution aiming at clarifying the role of software within the process and its environment. The patterns are illustrated on two real, non-trivial case studies.

1 Introduction

Process support is often advocated as a means for achieving higher-quality healthcare [8, 20]. In particular, medical *guidelines* capture evidence-based practices for handling specific diseases [13]; *clinical pathways* provide a patient-centric view of medical treatments involving multi-disciplinary teams, such as cancer treatments [28].

Models in this context enable the analysis, documentation and explanation of medical processes; they provide the basis for automated process support. Many languages and techniques are available for process modeling and analysis [17, 18], including UML activity diagrams [23], YAWL [31], BPMN [24], Little-JIL [34, 4, 5], and g-HMSC [9, 10].

Enactment techniques appear promising for process support in medical environments. Process *enactment* is commonly defined as the use of software to support the execution of operational processes [25, 12, 30]. In *model-driven* enactment, a high-level process model is used as input for execution support. An execution semantics of the modeling language must then be provided. For instance, YAWL has an execution

adfa, p. 1, 2011.

semantics in terms of Petri nets [31]. The Business Process Execution Language (BPEL) similarly complements BPMN while focusing on web service invocation [1].
Various process enactment schemes are available in medical environments, e.g., to dynamically maintain medical worklists, issue warnings or reminders in specific process states, schedule tasks competing for resources, or provide on-the-fly medical advice.
Our experience in a variety of medical environments suggests that specific model-driven enactment schemes are effective in specific contexts only. For example, one scheme may appear appropriate for supporting daily operations in a radiotherapy department while less appropriate for enforcing the clinical pathway for stroke treatment. In the former case, the model may focus on daily tasks to be handled by medical staff within a single department; in the latter case, the model may focus on highly time-critical tasks to be coordinated among multiple departments.
In order to better understand the fundamental nature of medical process enactment and integrate the diversity of possible enactment schemes, the paper introduces a conceptual framework for process execution and enactment. This framework is based on a goal-oriented, multi-view model of process execution that highlights the role of specific agents, including software, in executing medical processes. The model is parameterized on tasks; it separates the operational process being executed within some environment from the software supporting such execution. More specifically, this model integrates a goal model underlying process execution, a companion structural model relating all concepts involved in the goal model, and a behavior model derived from the goal model that highlights execution states and transitions on which software enactors may be anchored.
The proposed framework may be used for defining, comparing, classifying or integrating various enactment strategies. In particular, patterns are defined in this framework as common forms of process execution and enactment. The framework is intended for process analysts and software engineers to reason about the introduction of software enactors for process support.
The paper is organized as follows. Section 2 summarizes some background material used in the next sections. Section 3 describes our multi-view model for process execution and enactment. Section 4 analyzes generic assets used in medical task performance together with common patterns of process execution and enactment; these assets and patterns are obtained by instantiating the multi-view model on specific tasks. Section 5 briefly discusses two real medical case studies we were involved in that illustrate some of these patterns.

2 Background: Modeling Medical Processes

Guarded High-level Message Sequence Charts (g-HMCS) are used in the paper for modeling medical processes. The g-HMSC language is a simple flowchart-style formalism for modeling multi-agent processes involving *decisions* on process variables [9]. An *agent* is an active system component playing a specific role in the considered process. Agents cooperate to satisfy the medical objectives assigned to them [19]; they can be humans, devices or software.

The g-HMSC language is used for formal analyses while being close to the informal sketches provided by medical stakeholders [10]. As Fig. 1 suggests, a g-HMSC model is a directed graph with three types of nodes.

- A *task node* captures a process task, that is, a work unit performed by collaboration of agent instances involved in the process. It is represented by a box. The arcs connecting task nodes specify how these nodes must be composed sequentially.
  A task may be *refined* in another g-HMSC. A non-refined task (or *leaf* task) is specified through a scenario showing the temporal sequence of interaction events among agent instances. Leaf tasks are considered here as black-box execution units under responsibility of a specific agent. The latter is called task *performer*. When multiple agents are involved in the same task, available mechanisms for refining agents, tasks, and their underlying objectives should be used for capturing multiple performers [19]; lack of space prevents us from considering this further here.
- A *decision node* captures a process decision. The latter states specific conditions on process variables [10] for tasks along outgoing branches to be performed.
- *Initial* and *terminal* nodes represent the start and end of the (sub-)process.

A g-HMSC task may be annotated with additional information such as its precondition for application, its minimum/maximum duration, the resources required, and so forth [10].

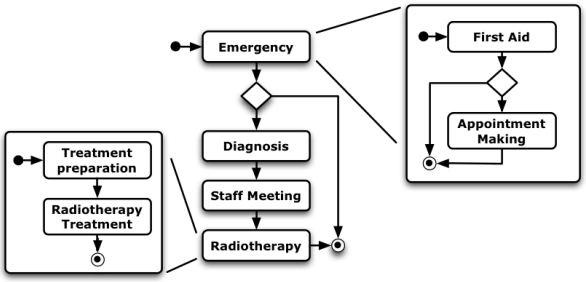

Fig. 1. A simple g-HMSC process model for cancer radiotherapy

3 Modeling Process Execution and Enactment

Section 3.1 introduces a generic goal model for process execution in a multi-agent setting together with a structural model interrelating the concepts involved in it. Section 3.2 discusses a corresponding behavior model highlighting execution states of process tasks together with transitions among those states. These transitions define the specific places at which software agents can be introduced for process enactment.

3.1 A goal model for process execution

This section aims at clarifying what a medical process enactor should actually be doing and why it should do so. A generic goal model is built for this; its instantiation provides the basis for defining execution strategies and enactment patterns.

A goal-oriented model integrates the intentional, structural, functional, and behavioral facets of the target system in a multi-agent setting [19]. It allows, among others, requirements on cooperating agents to be derived from high-level goals.

Fig. 2 (a) shows the goals (in parallelograms) underlying the execution of a medical process, how they are AND-refined into subgoals (through AND-arrows), and how they are assigned to agents (in hexagons). Fig. 2 (b) provides a structural model (as a UML class diagram) showing how the concepts appearing in the goal model are interrelated. Precise definitions for the goals in Fig. 2 are given in Fig. 3.

The root goal of process enactment states that "*all task instances must eventually be performed*". (As we capture process executions, we are mostly concerned with task instances; "task performance" is often used as a shortcut for "task instance performance". Fig. 2 shows a refinement-by-milestones [19] of this root goal:

- a first subgoal requires the need for performing a specific task to be known by its performer;
- once the need is known, the task shall eventually be performed.

The first subgoal is further refined. A *divide-and-conquer* refinement [19] is used:

- potential task performers shall be informed of the task performance need;
- one of these shall be selected as performer;
- agent selection and information shall not be undone.

The latter subgoal is set as an assumption to simplify matters. "Informed" and "selected" mean that if a task is performed then its performer knew that it was needed and that it was the one responsible for the task, respectively. The divide-and-conquer strategy is taken to cover various process execution scenarios, e.g. "selecting then informing" or "making all candidates informed and then selecting among them". In any case, a state must eventually be reached where the performer is both selected and informed.

The second subgoal of the root goal in Fig. 2 is refined as follows:

- the task performance shall first be started;
- it shall eventually be completed when started.

The goal model in Fig. 2 calls for further clarifications.

- The tasks to be performed there correspond to instances of *leaf* tasks in the process model; the structuring of the process model through task refinements has no impact on process executions.
- The goals of the generic model for process execution in Fig. 2 should not be confused with the objectives underlying process tasks. For example, the process model in Fig. 1 has underlying objectives such as *Achieve* [Treatment Prepared When Patient Registered] or *Maintain* [Patient Registered Before Radiotherapy]. The latter objectives are structured and refined in a specific goal model complementing the g-HMSC task model. Synergetic links exist between the treatment-specific and generic goal models; there are however not the primary focus of this paper.
- As specified in Fig. 2, the goal of a task being actually performed is assigned to the task performer. This requires the medical objective underlying the task to be *realizable* by the agent, that is, the agent must have the capabilities of *monitoring* the conditions to be evaluated and of *controlling* the conditions to be established according to this objective [19]. Agents playing the *Performer* role might be the tar-

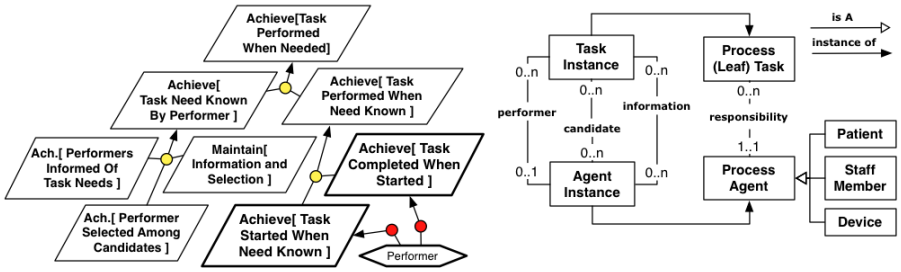

Fig. 2. (a) Generic goal model and (b) Structural model for process execution

Achieve [Task Performed When Needed]: Every task instance whose performance is needed shall eventually be performed by an instance of the responsible agent, called performer, under specific timing and resource constraints.

Achieve [Task Need Known By Performer]: For every task instance whose performance is needed, the performer shall eventually know this task instance is needed.

Achieve [Performers Informed Of Task Needs]: Every need for task performance shall eventually be known by at least one candidate performer.

Achieve [Performer Selected When Task Need]: For every task instance whose performance is needed, the actual performer shall eventually be selected among possible candidates.

Maintain Information and Selection]: Once selected a performer shall remain selected until the task is performed. Once informed a performer shall remain informed of the task performance need until the task is performed.

Achieve [Task Started When Need Known]: When the performer knows the need for performing a task instance, it shall eventually start that task so as to meet the corresponding timing and resource constraints.

Achieve [Task Completed When Started]: Every task instance whose performance has been started shall eventually be completed successfully.

Fig. 3. Specifications for the goals in Fig. 2 (a)

get patient, members of the medical staff (e.g., nurse, oncologist), medical devices (e.g., pump, radiotherapy machine) or available software (e.g., patient record system).

- The three other leaf goals in Fig. 2 should be further refined to specify how task performers and software enactors cooperate in order to satisfy them. Various alternative refinements lead to various enactment patterns, see Section 4.
- The model is deliberately kept simple for the purpose of this paper. It can easily be extended for capturing multiple task performers, through the agent/goal refinement mechanism [19], and exceptions, through obstacle analysis [19].

3.2 A behavior model for process execution

The various execution milestones identified in our generic goal model lead to the behavior model captured as a UML state diagram in Fig. 4. Three main execution states are identified.

– *Created*: Every task instance enters this state as soon as a new process instance is started. As a process may involve multiple decisions, some task instances might not be performed in practice even though they conceptually exist.

– *PerformanceNeeded*: Every task instance is in this state when its performance need is confirmed (in a way to be made further precise). This state is further decomposed in a way consistent with the goal model.
– *Performed:* Every task instance whose performance has been completed by its performer ends up in this state.

This behavior model is generic on tasks to be performed. Its transitions are therefore not labeled with events and/or actions; they are simply identified by a number. The guards there are derived from the goal model [19]; additional guards may be introduced when instantiating the model.

The state transitions in this behavior model yield the various places where software enactor agents may be introduced to drive the medical process. These transitions are discussed successively, focusing on *when* the transition gets fired and *how* enactment software may be involved. This paves the way to enactment patterns in Section 4.

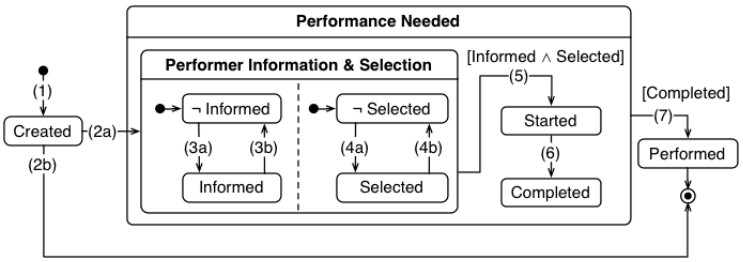

Fig. 4. States and transitions in the execution of a task instance

(1) *A task instance gets created as soon as a process instance is started*. Model-driven enactors typically use the process model to create task instances for further driving, thereby firing this transition.

(2a) *The need for task performance gets confirmed; the information and selection subprocesses of a task performer start accordingly.* This transition gets fired as soon as a process agent knows that performing a medical task is needed. The agent might be the patient deciding to go for consultation, the doctor prescribing some medical test, etc. In such cases, software enactors mostly help tracking corresponding needs. They might also help firing this transition by introducing effective *need identification means*.

(2b) *Task performance is not required; execution ends up immediately*. This transition accounts for task instances that conceptually exist but are not executed due to process decisions.

(3a) *The performer gets informed of the need to perform the task.* Various situations may correspond to this transition getting fired. The performer might be the one who identified the need –in which case he/she/it gets automatically informed. Alternatively, the agent might be informed by a specific request –such as a phone call, for example. Software enactors might help firing this transition by introducing effective *information means*.

(3b) *The performer gets unaware of the need to perform the task.* This transition is introduced to account for exceptional situations where, e.g., performers forget about their task commitments. Software enactors might help avoid this transition from getting fired by introducing effective *remembering means*.

(4a) *The performer gets selected among possible candidates.* This transition is fired as soon as the agent instance to actually perform the task is known. In simple situations with only one candidate or a predetermined selection result, the transition gets fired automatically without requiring any kind of support. Otherwise, software enactors might help firing this transition by introducing effective *selection means*.

(4b) *The selected performer gets unselected.* This transition is introduced to account for exceptional situations where, e.g., selected performers suddenly get unavailable –introducing the risk of tasks being not performed or performed too late. Enactors might be introduced to detect such situations and mitigate their consequences.

(5) *The task performance gets started under the necessary condition that the performer is both selected and informed*. In our framework, the responsibility of starting a task is assigned to its performer. A *trigger condition* should capture when the transition gets fired, e.g., "as soon as the performer is available" or "at some predefined moment". Software enactors may help performers here.

- As seen before, effective *remembering* and *selection means* help keeping the guard satisfied for tasks whose performance is needed.
- *Prioritizing means* may help selecting which task instances to perform among those having their transition currently enabled.

(6) *The performer completes task performance*. Here again, the responsibility for task completion is assigned to the performer. Software enactors may assist here in achieving the task's post-condition (e.g., automated checklists) or in detecting task instances whose performance has started but has not been completed yet.

(7) *As the task has been completed its performance is no longer needed.* This transition captures that the performing agent gets informed that the completed task is no longer needed. In many cases, this transition is automatic; it results from the performing agent being aware of task completion. When software enactors are involved, the transition is required for enactors to be informed of tasks being no longer needed, e.g., to remove them from task lists or to determine subsequent tasks to be performed. This may prove challenging as a feedback loop is required from performers to enactors [7, 3]. An explicit feedback requires performers to inform enactors; an implicit one uses some *monitoring means*.

4 Patterns of process execution and enactment

The intentional, structural and behavior models of process execution in Section 3 allow us to identify patterns commonly found for getting tasks performed under specific timing and resource constraints. When software enactors are introduced, such patterns may be called *enactment* patterns. An execution/enactment pattern relies on

specific task performance *assets*. Section 4.1 reviews such assets by instantiation of the concepts introduced in Section 3.3 –such as *information means*, *remembering means* or *trigger condition*. Assets are not necessarily independent from each other; patterns for using them together may therefore be identified. Section 4.2 lists various examples of how assets may be combined in such patterns.

4.1 Task performance assets

The firing means for the successive state transitions in Section 3.3 are instantiated with a particular focus on software enactors. The process in Fig. 1 is used as a running example for illustrating various cases.

Need identification means: *How is a task known to be needed?*

Symptoms or accidents may cause patients to enter a medical process. In Fig. 1, the Surgery task might be needed because of a patient feeling some specific pain. The need for performing a task may also result from a medical decision or a diagnosis made by some medical actor. For example, after the First Aid task in Fig. 1, cancer suspicion may raise the need for further diagnosis with appointments being made accordingly.

Software-aided diagnosis therefore falls into this category of process enactment [16]. *Patient monitoring systems* inside or outside the hospital might be seen as enactors too for the same reason [3, 15, 32].

Another source for identifying task needs is provided by medical guidelines. *Rule-based* or *procedural process models* formalizing guidelines may be used as assets for driving processes through dedicated enactors [22]. In such cases, the execution semantics of the modeling formalism prescribes how the software can infer which task is needed in the current process state.

Information means: *How does the performer know that the task is needed?*

Various information means are available for informing potential performers of tasks to be performed.

- Direct or indirect *requests* (e.g. phone calls, messages) are a usual way of informing performers of tasks to be done. For example, the Appointment Making task in Fig. 1 might simply consist in asking someone at the department's welcome desk.
- *Calendars* are another means for keeping track of tasks to be performed. For cancer treatment, for example, nurses or assistants may fill in the radiotherapist's e-calendar with specific dates for the Diagnosis task (see Fig. 1). The patient may also keep track of the dates of her Radiotherapy treatments in her own calendar.
- *Registration lists* are sometimes used for recurrent task instances. A registration list might be filled in by nurses or assistants to inform radiotherapists and oncologists of the patients to be discussed at the next multidisciplinary Staff Meeting task instance (see Fig. 1).
- *Worklists* may also be used for keeping track of tasks to be performed and informing agents about these. They might be physical ones or managed by software. For example, the medical staff might know that a new Emergency task instance in Fig. 1 is needed by looking at the physical waiting queue. The Treatment Prepara-

tion task might rely on a digital worklist for organizing the preparation of pre-planned radiotherapy treatments.

Automated worklists are frequently mentioned in the literature; they originate from agenda management systems [14, 21]. Most workflow management systems actually depend on such lists for enacting process models, see e.g. [34, 2, 33].

Remembering means: *How does the performer remember that the task is needed?*

Most information means may also be used as remembering means –in particular, calendars, registration lists and worklists.

- *Reminders* are also commonly used in medical practice [16]. They can be easily automated for tasks whose performance time is known in advance –e.g., from the Diagnosis appointment known from the medical e-agenda (see Fig. 1). When effective *identification* and *information means* are available, more complex reminding schemes can be implemented. For example, a reminder to participate to the next Staff Meeting task might be automatically sent to the Radiologist only under the condition that at least one of her patients appears on the registration list.

Selection means: *How is the performer selected among candidates for the task?*

In situations where the assignment of tasks to performer instances is not necessarily obvious, various selection strategies are available.

- The *first-available-performer* strategy accounts for situations where a pool of performer instances process tasks as the flow is going. This strategy is commonly used when a worklist is used as information means –e.g. as in the Emergency task, or in the Treatment Preparation task that might involve a pool of physicists acting according to a digital worklist (see Section 5.1).
- *Static allocation* assigns tasks to performers on a pre-established basis. For example, chemotherapy treatments might be organized according to a pre-established assignment of beds to nurses.
- *Dynamic allocation* assigns tasks according to resource availability and other constraints. Performer instances do not decide which task instances they perform. Software may be used here for enactment and sometimes for task performance. The automated scheduling of Radiotherapy Treatments according to available treatment machines is an example of this.

Selection means are not intended to make dynamic choices among agent instances based on their respective role or capabilities; this is achieved at modeling time through the introduction of different agents with specific roles, capabilities, and responsibility assignments [19].

Prioritizing means: *How are task instances selected for performance?*

While *selection means* prescribe how performers are selected, *prioritizing means* prescribe in what order the latter perform tasks among those needed. Prioritizing means are commonly used when task instances compete for resource availability.

- *First-In-First-Out*: This static strategy is often used in combination with pure queue-based worklists –see, e.g., the Appointment Making task in Fig. 1. Registration lists sometimes implement this strategy as well.

- *Highest-Priority-First:* This dynamic strategy is also frequently used in combination with worklists and registration lists –e.g., to account for urgency in the Emergency task or in Staff Meeting discussions. Software enactors might be introduced here as well. For example, instances of the Treatment Preparation task in Fig. 1 might be dynamically prioritized in a worklist according to known dates for the corresponding Radiotherapy Treatment.
- *Automated Scheduling:* Prioritization with dynamic performer allocation under complex constraints may require an *online* or an *offline* scheduler. This software enactor computes what tasks are to be performed first, either on the fly (online) or ahead of time (offline).

Trigger conditions: *What makes tasks getting started?*

In correlation with the *information*, *remembering*, *selection* and/or *prioritizing* means, task instances occur at a specific time and/or for specific reasons. Provided the required resources are available and the task's precondition is met, the following conditions may trigger task performance.

- *As-Soon-As*: When a request or a worklist is used as information means, tasks are sometimes performed as soon as the selected performer instance is available. This might for example be the case for the Appointment Making task in Fig. 1.
- *Highest-Priority*: When prioritization means are used, a task may start as soon as it gets the highest priority. The First Aid and Treatment Preparation tasks in Fig. 1 are typical examples of this.
- *Fixed-Time*: Many medical tasks simply occur at some fixed, predetermined time. In Fig. 1, the Diagnosis task might occur according to a predetermined appointment. Radiotherapy Treatments similarly follow a pre-established schedule. Staff Meeting instances generally occur at a specific time slot every week.

4.2 Combining performance assets: process execution/enactment patterns

Fig. 5 shows a number of typical patterns obtained when performance assets are combined. Those patterns provide a way of explaining why tasks are actually performed in practice and how software enactors may support process execution. The list of identified patterns is obviously not exhaustive.

Request-driven: This pattern refers to situations where a task is performed on request, as soon as its performer is available, and in a first-in-first-out order. It might capture how tasks are performed at a welcome desk or how requests issued by bedridden patients are handled. The Appointment Making task in Fig. 1 follows this pattern.

Appointment-driven: This pattern refers to tasks such as Diagnosis or Consultation. For such tasks, calendars are used by performers to agree and remember fixed-time appointments. Most often, the performer is statically known as in the case of Consultation with a specialist. Support for e-calendars accessible to medical staff is then an effective form of enactment.

		Request-driven	Appointment-driven	Meeting-driven	Priority-driven	Resource-driven	Scheduling-driven
Information	Request	x					
	Calendar		x				
	Registration list			x			
	Worklist				x	x	
Selection	First available						
	Static allocation		x				
	Dynamic allocation						x
Prioritizing	FIFO	x					
	Priorities				x		
	Scheduling						x
Trigger	As soon as	x				x	
	Highest priority				x		
	Fixed time		x	x			

Fig. 5. Combining assets: process execution and enactment patterns

Meeting-driven: This pattern refers to tasks such as Staff (see Fig 1) or Support Group Meetings. Such meetings occur at a fixed, recurrent time –e.g., once a week. Registration lists are used to track which patients must be discussed or need be present. Automating such lists and the registration process results in effective information and remembering means.

Priority-driven: Automated worklists correspond to a well-known kind of model-driven process enactment. An associated pattern uses priorities to decide which task instances must be performed first among those in the worklist. This pattern is compatible with all assets for performer selection. However, it assumes that performers are immediately available after having performed a task. The pattern is therefore better suited for automated performers such as medical devices. Hospital pharmacy automation provides a good example of use.

Resource-driven: This pattern is a variant of the previous one. It also relies on worklists as information means. Task performance is not driven by priorities here but by the availability of scarce resources such as the performer itself; the tasks are performed as soon as the required resources get available. The pattern is compatible with various *prioritizing* and *selection* means. The Treatment Preparation task in Fig. 1 is an example of application (see also Section 5.1).

Scheduling-driven: This pattern is frequently used when multiple instances of a task compete for scarce and/or costly resources. The dynamic allocation of those resources using available scheduling techniques is an effective enactment strategy in such cases. The scheduling of sessions on radiotherapy machines typically relies on this pattern (see Section 5.1). The determination of patient appointments under complex resource and time constraints may also benefit from this pattern (see Section 5.2). The scheduling may involve real-time slots decided ahead of task performance. In such cases, the pattern also relies on calendars and fixed-time appointments.

5 Case studies

This section briefly reports on two quite different instantiations of the multi-view model in Section 3 and patterns in Section 4. These instantiations capture the effective

enactment of complex processes in real medical environments. Section 5.1 refers to a software-managed daily worklist for the preparation of radiotherapy treatments. Section 5.2 outlines a tool for automated scheduling of appointments for chemotherapy treatments. Both examples are taken from real medical projects we were involved in at the UCL university hospital in Brussels.

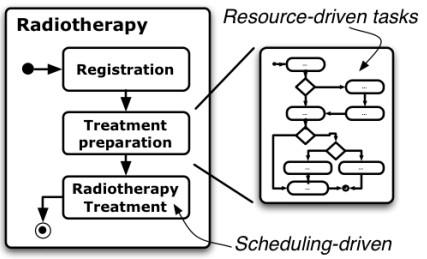

Fig. 6. Simplified radiotherapy process

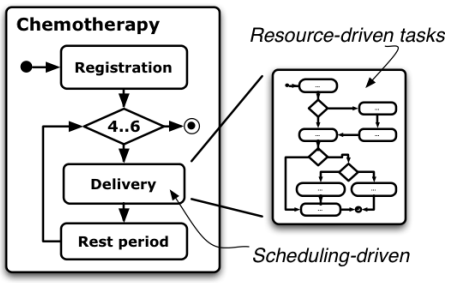

Fig. 7. Simplified chemotherapy process

5.1 Automated worklist for preparing radiotherapy treatments

Fig. 6 shows a simplified process model for the preparation of radiotherapy treatments. A patient enters the process once her cancer has been confirmed by earlier diagnosis. The process starts with an administrative Registration task. In particular, the dates of the treatment on a specific therapy machine are scheduled at registration time. The reason is that radiotherapy machines are heavily contentious resources. A *scheduling-driven* enactment pattern is therefore used for determining the treatment dates and machines allocated for the Radiotherapy Treatment task.

This yields a specific problem for the Treatment Preparation task. For a given patient, radiotherapy preparation has a strict deadline of *14* days due to specific constraints not discussed here. The Treatment Preparation task is decomposed into subtasks handled by medical staff. (These subtasks are not detailed here for lack of space.) None of these subtasks requires the patient to be present; some require a physicist, others an assistant, and others a radiotherapist. For a given set of patients to be treated, with corresponding process instances, a daily problem is to avoid rescheduling radiotherapy sessions dynamically (to save costs). The preparation must therefore be organized so as to meet all deadlines and constraints.

An effective enactment solution consists of using an automated worklist for the Treatment Preparation subtasks, according to the *resource-driven* pattern. A software enactor implementing this pattern is used daily at the radiotherapy department of the UCL university hospital.

- The *information means* are given by the automated worklist itself. All tasks to be performed are known by the multi-disciplinary team. The worklist is however filtered by medical role.
- Various *selection means* are possible with this pattern. Here, *static allocation* on a per-patient basis is used for tasks handled by medical assistants. In contrast, physicists are selected according to the *first-available-performer* strategy.

- The actual performance of tasks is driven by resource availability. In the ideal case, a subtask is performed as soon as a performer is available (*trigger condition*) provided the task gets the highest priority (*prioritizing*). Priorities are carefully determined according to: (a) the deadline imposed by scheduling on treatment machines, and (b) statistics about the time taken by each subtask.
- To push the currently implemented approach further, a prototype tool was developed for *model-driven* enactment. A g-HMSC model of the Treatment Preparation subprocess was used to fill in and dynamically maintain the worklist automatically (*identification means*).

In terms of the state diagram in Fig. 5, transitions (2a), (2b) and (7) require some dedicated treatment through collaboration between medical staff and the software enactor. Transition (7) requires medical staff to explicit mark tasks as done in the worklist. This allows the enactor to remove these and replace them by successor tasks according to the process model –thereby firing transitions (2a) and (2b).

5.2 Scheduling chemotherapy treatments

A process model for simple chemotherapy-based cancer treatment is sketched in Fig. 7. While apparently similar to the previous one, the enactment solution we are developing is fairly different here.

Cancer is treated through a series of several sessions (or *cycles*) of chemotherapy treatment over a few months. Various treatment plans are available according to the kind of cancer, the patient's age, the drugs to be used, and so forth. For instance, a FEC chemotherapy cycle for treating breast cancer takes about 21 days [26]. On the first day of each cycle, the patient goes to hospital for an injection of FEC chemotherapy drugs (Delivery task in Fig. 7). Then, she has no chemotherapy for the next 20 days (Rest Period in Fig 7). Four to six similar cycles are followed. Depending on the treatment plan, the Delivery task takes from a few minutes to a few hours. Subtasks include a consultation with an oncology assistant, drug preparation by the hospital's pharmacy, and drug injection by a nurse in some bed (to be determined). These Delivery subtasks are all driven by resource availability.

Unlike the previous case study, the problem here is not to help with the organization of those subtasks *per se*. Instead, the date of each Delivery task instance should be determined for the entire patient population –taking into account that different patients follow different treatment plans requiring different resources such as nurses, assistants, or beds; the latter are available in limited number.

The date of the first treatment is determined during the Registration task. Dates for the next treatments are determined at the end of each Delivery task. Some flexibility is provided to the patient for picking up those dates. For the treatment to be successful, however, the length of each cycle must be kept as close as possible to the one prescribed in the treatment plan.

A software enactment solution is currently being developed in our research group in collaboration with a medical team at the day care department of the UCL university hospital. It consists in scheduling drug Delivery task instances among patients over time so as to meet various types of constraints (detailed hereafter).

– An electronic calendar is used as *information means*. It lets both medical staff and patients know when each Delivery instance takes place.
– *Prioritizing* is achieved using constraint-based local search technology [11]. Our scheduler generates Delivery dates for each patient so as to meet the following:
  - the corresponding process model, in particular, the time bounds on the various treatment cycles;
  - resource availability constraints;
  - a target criterion on a safety-critical process quality indicator. More specifically, the *Relative Dose Intensity* (RDI) indicator captures how close the dose actually delivered to a patient over time is to the optimal dose intensity. Roughly, the higher the RDI the closer the actual treatment is to the optimal treatment plan prescribed by the evidence-based process model.

The scheduler continuously computes "best" date proposals and fills in the e-calendar. The proposal for the next delivery date is accepted or adapted by the patient at the end of each Delivery task instance, in agreement with the medical assistant and nurse. When the proposed dates are changed in the calendar, the impact of the change on the resulting RDI is shown as feedback to patients, assistants and nurses to possibly warn them about the consequences of the change on the treatment effectiveness.

6 Conclusion

The paper presented a goal-oriented multi-view model for reasoning about medical process execution and enactment. Task performance assets and process execution/enactment patterns were identified by instantiation of this generic model. The objective was by no means to define yet another process modeling or execution language. Rather, the proposed framework provides abstract support for process analysts and software engineers to reason about medical process executions and anchor software enactors on the process. Even though the g-HMSC process modeling language is incidentally used, our conceptual framework does not really depend on a particular modeling language. Similarly, it does not presuppose any particular execution semantics.

The richness, variety and complexity of medical environments provide evidence that human-intensive processes are first and foremost executed on the field [7]. For software enactors to be effective, a fine understanding of the process environment appears a prerequisite. Experience with execution/enactment patterns suggests that they are worth reusing. Further understanding of the nature of medical process execution should yield more patterns and increase their effectiveness.

As mentioned before, the simple conceptual framework in this paper may easily be extended to capture multiple agents cooperating to the same task; the available mechanisms for refining goals, agent, assignments and tasks may be used for this [19, 9, 10]. Resources and their capabilities should also be brought explicitly into the framework [27]. Exceptions need to be more thoroughly considered [29]; currently, only cases of tasks being forgotten or performers being unavailable are covered. Systematic obstacle analysis [19] against leaf goals in our multi-view execution model would integrate exceptional situations where tasks are cancelled, resources are unavailable,

task pre- or postconditions are violated, and so forth. This would result in a more comprehensive and robust behavioral model with new corresponding states and transitions. New software enactors could then be plugged in for those transitions, and new patterns identified accordingly. For instance, anchoring the detection of deviations from the normal process [6] would require transitions capturing tasks being forgotten or cancelled.

Process modeling requires the underlying process goals, expectations on human agents, and requirements on medical devices to be structured in a process-specific goal model. The execution/enactment patterns in practice depend on such goals and their potential obstacles. Future work should therefore be devoted to the synergetic links between the process model and its companion goal model on the one hand, and their relation to the generic execution framework and enactment patterns outlined here on the other hand.

Acknowledgement. We wish to thank P. Scalliet and M. Coevoet for providing us with details about their radiotherapy worklist software outlined in Section 5.1. Many thanks are also due to R. De Landtsheer, F. Roucoux, Y. Guyot, C. Ponsard and Y. Humblet for their collaboration in designing the scheduling engine in Section 5.2. This work was supported by the Regional Government of Wallonia (PIPAS project Nr. 1017087).

7 References

1. T. Andrews et al, Business Process Execution Language for Web Services, Version 1.1, May 2003; *see also* OASIS Standard WS-BPEL 2.0.
2. R. Anzböck, R. and S. Dustdar, "Modeling and implementing medical web services", *Data & Knowledge Engineering*, 55(2), 2005, 203-236.
3. S. A. Behnam and O. Badreddin, "Toward a Care Process Metamodel for Business Intelligence Healthcare Monitoring Solutions." *Proc. of SEHC'2013: 5th Intl. Workshop on Soft. Eng. in Health Care*, 2013, 79-85.
4. B. Chen, G. S. Avrunin, E. A. Henneman, L. A. Clarke, L. J. Osterweil, P. L. Henneman, "Analyzing medical processes", *Proc. ICSE'2008: 30th Intl. Conf. on Software Engineering*, ACM-IEEE, 2008, 623-632.
5. S. Christov, B. Chen, G. S. Avrunin, L. A. Clarke, L. J. Osterweil, D. Brown, L. Cassells, and W. Mertens, "Rigorously defining and analyzing medical processes: an experience report". In H. Giese (Ed.): *MODELS 2007 Workshops*, Springer-Verlag, 2008, 118-131.
6. S. C. Christov, G. S. Avrunin and L. A. Clarke, "Considerations for online deviation detection in medical processes." *Proc. of SEHC'2013: 5th Intl. Workshop on Soft. Eng. in Health Care*, 2013, 50-56.
7. L.A. Clarke, L.J. Osterweil, and G.S. Avrunin, "Supporting human-intensive systems", *Proc. of FSE/SDP workshop on Future of Soft. Eng. Research*, 2010, pp. 87-92.
8. P. Dadam, M. Reichert and K. Kuhn, Clinical workflows—the killer application for process-oriented information systems?, Springer London, 2000, pp. 36-59.
9. C. Damas, B. Lambeau, F. Roucoux and A. van Lamsweerde, "Analyzing critical process models through behavior model synthesis, *Proc. ICSE'2009: 31st Intl. Conf. on Software Engineering*, Vancouver, 2009, 441-451.
10. C. Damas, B. Lambeau and A. van Lamsweerde, "Analyzing Critical Decision-Based Processes", *IEEE Transaction on Software Engineering*, Vol. 40 No. 4, April 2014, 338-365.
11. R. De Landtsheer and C. Ponsard, "Oscar.cbls: an open source framework for constrain-based local search", 27th ORBEL Annual Meeting, Kortrijk, February 7-8 2013.

12. A. Finkelstein, J. Kramer and B. Nuseibeh (Eds.), *Software Process Modelling and Technology*. Research Studies Press Ltd., Taunton, UK, 1994.
13. C. Gordon, M. Veloso and the PRESTIGE Consortium, "Guidelines in healthcare: the experience of the PRESTIGE Project", Medical Informatics Europe, IOS Press, 1999.
14. M. Heisel, "Agendas – a concept to guide software development activities", *Proc. of the IFIP TC2 WG2: 4th working conference on Systems implementation, languages, methods and tools*, Chapman & Hall, 1998, 19-32.
15. J. C. Hou, "Pas: A wireless-enabled, sensor-integrated personal assistance system for independent and assisted living." *High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability*, 2007.
16. M. E. Johnston, K. B. Langton, R. B. Haynes and A. Mathieu, A. "Effects of computer-based clinical decision support systems on clinician performance and patient outcome: a critical appraisal of research." *Annals of internal medicine, 120(2)*, 1994, 135-142.
17. G.T. Jun, J.R. Ward and Z. Morris, "Health care process modelling: which method when?", *International Journal for Quality in Health Care*, Vol. 21 No.3, 2009, 214-224.
18. S. Kaiser and S. Miksch, *Modeling Computer-Supported Clinical Guidelines and Protocols: A Survey.* Vienna Univ. Technology, Report Asgaard-TR-2005-2, 2005.
19. A. van Lamsweerde, Requirements Engineering: From System Goals to UML Models to Software Specifications. Wiley, 2009.
20. R. Lenz and R. Manfred, "IT support for healthcare processes", *Business process management*, Springer-Verlag, 2005. 354-363.
21. E. K. McCall, L. A. Clarke and L. J. Osterweil. "An adaptable generation approach to agenda management.", *Proc. ICSE'1998: 20th Intl. Conf. on Soft. Eng.*, 2008, 282-291.
22. J. Mathe, J. Sztipanovits, M. Levy, E.K. Jackson and W. Schulte, "Cancer Treatment Planning: Formal Methods to the Rescue". *Proc. of SEHC'2012: 4rd Intl. Workshop on Soft. Eng. in Health Care*, Zurich, Switzerland, 2012.
23. OMG, UML 2.0 Superstructure Specification, 2003.
24. OMG, Business Process Modeling Notation, v1.1, 2008.
25. L. J. Osterweil, "Software processes are software too", *Proc. ICSE '87: 9th Intl. Conf. Software Engineering*, ACM-IEEE, 1987, 2-13.
26. M. C. Perry, *The chemotherapy source book*, Lippincott Williams & Wilkins, 2008.
27. M.S. Raunak and L. J. Osterweil. "Resource Management for Complex, Dynamic Environments" , *IEEE Transactions on Software Engineering* 39.3, 2013, 384-402.
28. M. Renholm, H. Leino-Kilpi, T. Suominen, "Critical pathways: a systematic review", *Journal of Nursing Administration* 32(4), 2002,196-202.
29. B. Staudt Lerner, S. Christov, L.J. Osterweil, R. Bendraou, U. Kannengiesser, and A. Wise, "Exception handling patterns for process modeling", *IEEE Transactions on Software Engineering* Vol. 36 No. 2, pp. 162-183, 2010.
30. W. Van Der Aalst, A. Ter Hofstede, and M. Weske, "Business process management: A survey", *Business process management*, Springer-Verlag, 2003. 1-12.
31. W. van der Aalst et al., "YAWL: yet another workflow language", *Information Systems* 30(4), 2005.
32. Q. Wang, W. Shin, X. Liu, Z. Zeng, C. Oh and B.K. AlShebli. "I-Living: An Open System Architecture for Assisted Living". *In SMC*, 2006, 4268-4275.
33. M. Westergaard and F. M. Maggi, "Declare: A Tool Suite for Declarative Workflow Modeling and Enactment", *BPM (Demos)*, 2011, 820.
34. A. Wise, A. G. Cass, B. S. Lerner, E. K. McCall, L. J. Osterweil, and S. M. Sutton, "Using Little-JIL to coordinate agents in software engineering", *Proc. ASE'2000: Automated Software Engineering Conference*, Grenoble, IEEE, pp. 155-163, 2000.