Generating Process Models in Multi-View Environments

Christophe DAMAS, Bernard LAMBEAU and Axel VAN LAMSWEERDE[1]

Université catholique de Louvain, Louvain-La-Neuve, Belgium
axel.vanlamsweerde@uclouvain.be

Abstract. The role of high-quality models is increasingly recognized for driving and documenting processes in safety-critical domains such as medical processes. In general, the environment of a complex process has to deal with multiple views referring to different concerns, agents, resources, locations, and so forth. The variety of needs along those views calls for multiple, complementary and consistent facets of a composite process model, where each facet addresses a specific view. Building multi-view process models is in our experience hard and error-prone. To address this challenge, the paper describes formal operators for composing process model facets in a coherent way and, conversely, for decomposing process models into specific facets that abstract from details irrelevant to a specific view. These operators are grounded on the formal trace semantics provided by our process language and its supporting analysis toolset. The paper shows how these operators are automated and reports on their use in a real case study of model construction for a particle-therapy center.

Keywords. Process modeling, model views, model transformation, process analysis, model verification, model synthesis, safety-critical workflows.

Introduction

Human-intensive systems involve complex cooperation processes among software applications, devices, and people, where the human contributions require domain expertise and have a significant impact on the success or failure of the overall mission [3, 7]. Such systems are in general safety-critical as exemplified by flight control, disaster management, or medical process support systems. Errors in medical systems, in particular, are known to cause more deaths yearly than the equivalent of one civil aircraft crashing every day.

Human-intensive processes should be captured through adequate models for effective software support [3, 10, 13]. Such models typically capture the control flow among tasks performed by the agents forming the considered system.

In general, human-intensive processes take place in complex environments covering a great variety of different aspects –refering to, e.g., different agents from different groups (such as organizational departments), different types of resources, different locations, and so forth. One process might be only concerned with the tasks performed by a specific subset of agents (e.g., a department workflow). Another might be a sub-process to be followed by a specific subset of process instances only. Yet

[1] Corresponding Author.

another might focus on tasks where specific resources are used. A composite process can thus be captured through different views along different levels of detail and different aspects to focus on.

To make this important motivating point further explicit, let us consider a medical process for cancer treatment. Different views on it may be captured through different workflows (see Fig. 1).

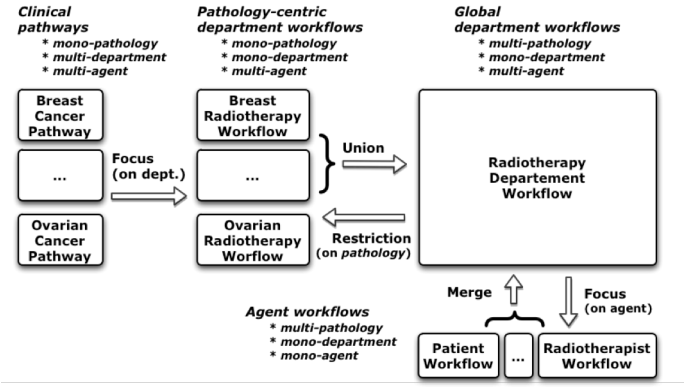

Figure 1. Variety of processes and model generation operators [11]

- A *clinical pathway* provides a multi-disciplinary view of the treatment required within a specific class of patients having the same pathology [30]. For example, we might consider the clinical pathway for treating ovarian cancers. The treatment process involves multiple agents cooperating across multiple medical departments.
- A *pathology-centric department workflow* shows the various decisions and tasks to be performed within a single department for a specific pathology. In our example, we might consider the workflow for treating ovarian cancers in the radiotherapy department.
- A *department workflow* provides a global view of the various decisions and tasks within a specific department. The process now refers to any patient treated in this department. In our example, we might be interested in the workflow of the radiotherapy department for any cancer being treated there.
- An *agent-centric workflow* focuses on the decisions and tasks of a specific agent. For example, the workflow for treating ovarian cancers as seen by the patient includes only those tasks in which the patient is involved. It provides a good basis for explaining her the treatment.

Such a simple workflow typology does obviously not cover all possible views one might be interested in. For instance, a mono-pathology, mono-agent and mono-department workflow might be worth considering; it is not shown in Fig. 1. Other possible aspects might include medical resources, patient characteristics such as age or gender, and so forth. For example, a "reminding workflow" might show only those tasks where the patient is responsible for taking specific drugs.

In such a multi-view setting, it won't make much sense to build a specific model for every process view of interest. Our objective instead is to generate models along specific views from others –and, in particular, to generate a composite model from partial views associated with specific aspects.

To achieve this, [11] informally introduced a general approach based on operators for composing or decomposing process models.

- *Composition* operators produce a composite model from separate ones in a semantically coherent way. Among these, the *Union* operator produces a model containing all paths that are valid in the input models and only those. The *Merge* operator produces a model covering all admissible interleavings of the concurrent input models.
- *Decomposition* operators produce specific views from a composite model in a semantically coherent way; each view abstracts from details irrelevant to it. The *Restriction* operator produces a model whose paths are followed by a subclass of process instances only. The *Focus* operator produces a model whose tasks form a subset of the set of tasks captured by the input model.

Composition/decomposition operators are aimed at supporting model construction, analysis, documentation and enactment.

- *Model construction*. A complex multi-view process model may be built by composition of view-specific fragments. In Fig. 1, the multi-pathology model for a department workflow might be obtained by application of the *Union* operator on all pathology-centric workflows managed by the department.
- *Model analysis*. Checks for desired properties may be performed manually by domain experts or automatically through formal analyses [3, 10, 12].
  - Manual inspections prove much easier when dedicated views are provided on the process model. Details relevant to a specific expert are obtained by application of the *Focus* operator on a specific department or agent, or by application of the *Restriction* operator to specific process instances.
  - Formal property checks may be applied to smaller models obtained by application of *decomposition* operators, for increased efficiency and easier understanding of counterexamples, or to larger models obtained by application of *composition* operators, for checking desired properties at a more global level.
- *Model documentation*. Any documentation generated from a process model should only cover those aspects of the process that are specifically relevant to the target audience. The *Restriction* and *Focus* operators may be applied to remove irrelevant aspects.
- *Model enactment*. The execution of concurrent process models considered in isolation might violate resource limitation constraints. Sound *composition* operators provide a means for handling this.

This paper expands the overall approach informally introduced in [11] with three main contributions.

- *Formal techniques* allowing those model composition and decomposition operators to be fully automated. In passing, a new variant of the *Merge* operator is introduced which proves frequently needed in practice.
- A *tool* implementing those formal techniques, integrated in our current formal toolset for process model analysis [12].
- An *evaluation* of our approach on a real, complex case study suggested by an industrial partner for orchestrating treatment processes in a particle-therapy center.

The language of guarded high-level message sequence charts (g-HMSC) is taken as process formalism for automating our operators. The g-HMSC language is a simple flowchart-like formalism for modeling multi-agent processes involving decisions on process variables [10, 12]. The language has a formal trace semantics defined in terms of guarded labelled transition systems (g-LTS), the latter having a trace semantics defined in terms of labelled transition systems (LTS) [10, 26, 27]. Automating g-HMSC composition/decomposition operators eventually reduces to g-LTS manipulations.

The paper is organized as follows. Section 1 summarizes some background material on the g-HMSC and g-LTS formalisms together with the different types of analysis being currently supported. Section 2 describes the various techniques for computing the *Union*, *Restriction*, *Focus*, and *M*erge of process models. Section 3 details basic common g-LTS manipulations involved in these techniques. Section 4 suggests how useful macro-operators can be defined by combining those composition and decomposition operators. Section 5 briefly discusses implementation aspects of the proposed techniques in our current toolset. Section 6 reports on the evaluation of our approach on a real, challenging modeling problem. Section 7 reviews some relevant related work before concluding.

1. Modeling and Analyzing Decision-based Processes

The g-HMSC language is a simple flowchart-style formalism for modeling multi-agent processes involving decisions on process variables. It is intended to be used by process analysts while being close enough to the informal sketches provided by medical experts [20]. The language has a formal semantics to enable different types of analysis [10, 12]. Fig. 2 shows g-HMSC model fragments for treating ovarian and endometrial cancers, respectively. These examples are simplified for explanation purposes. Roughly, the processes consist of a diagnosis task, a staff meeting and a surgery task. The choice of a specific surgical intervention (coelioscopy, laparotomy or lymphectomy) depends on the type of cancer and on a possible invasion of lymph nodes (N_+). For endometrial cancers with lymph node invasion, additional examinations should also be performed; they consist of a biopsy followed by anatomo-pathological analysis.

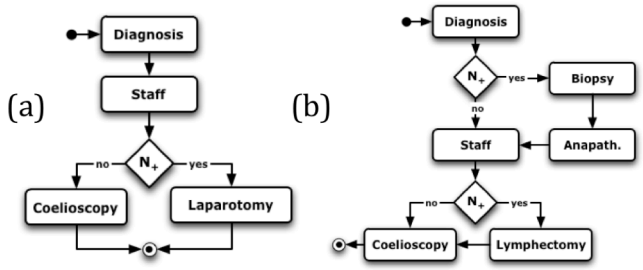

Figure 2. g-HMSC models for treating (a) ovarian and (b) endometrial cancers

1.1. Process Models as Guarded hMSCs

Guarded High-Level Message Sequence Charts (g-HMSCs) capture *decision-based processes,* that is, processes where decisions relying on the state of the process environment regulate the nature of subsequent tasks and their composition. For example, the choice of performing a coelioscopy or a laparotomy in Fig. 2a is the outcome of a medical decision relying on the invasion of lymph nodes (N_+).
A *g-HMSC model* is a directed graph with three types of nodes [10, 12].

- A *task node* captures a process task, that is, a work unit performed by collaboration of agent instances involved in the process. A g-HMSC task may be refined into another g-HMSC or in a message sequence chart (MSC) [19] showing the required sequence of interactions among the involved agent instances. A task is represented by a box. Outgoing arcs prescribe how the connected nodes must be sequentially composed.
- A *decision node* captures a process decision. A decision is characterized by a set of guards. Each *guard* is associated with a specific outgoing branch; it specifies the condition for the tasks along this branch to be performed. Guards are Boolean expressions on process variables (defined hereafter). A decision is represented by a diamond with a guard labeling each outgoing branch of the decision. In simple cases with two branches only, the guard expression may be moved up inside the decision node with ‘yes’, 'no’ labels attached to the corresponding branch.
- *Initial* and *terminal* nodes represent the start and end of the (sub-)process, respectively.

The *process variables* appearing in guards at decision nodes of a g-HMSC model dictate which paths are to be followed in specific process instances. They get their values from the occurrence of specific events; no explicit assignment is needed which makes g-HMSC models much simpler. Process variables may be task-related fluents or environment tracking variables.

- *Fluents* encode task occurrences. A *fluent* FL is an atomic proposition defined by a set $Init_{FL}$ of initiating events, a set $Term_{FL}$ of terminating events, and an initial value $Initially_{FL}$ that can be *true* or *false* [16].
- *Tracking variables* approximate unobservable environment quantities as accurately as possible through some observable counterpart [12]. When a tracking variable appears in a guard, the decision is based on its current value rather than the actual, unobservable counterpart. In Fig. 2, the Boolean variable N_+ tracks the fact that the patient’s lymph nodes are invaded.

A g-hMSC may have an *initial context condition* C_0 constraining the acceptable initial values of process variables. In Fig. 2a, the initial context condition is $C_0 = Ovarian$. This means that the process only applies for treatment of patients suffering from ovarian cancer; the tracking variable *Ovarian* is true in the initial state for any patient following this process.

A g-HMSC task may be annotated with optional information such as its applicability precondition, its minimum/maximum duration, its cost, the agents involved in its performance, the resources needed, and so forth. Such annotations are

used for specific types of analysis. For example, Table 1 lists the agents involved in tasks from Fig. 2.

The g-HMSC language has a formal trace semantics defined in terms of guarded labelled transition systems (g-LTS), the latter having a trace semantics defined in terms of LTS [10].

Task	Agents involved
Diagnosis	{oncologist, radiologist, patient}
Staff	{oncologist, radiologist, surgeon, anesthetist, pathologist}
Biopsy	{surgeon, nurse, patient}
Anapath.	{pathologist}
Coelioscopy	{surgeon, anesthetist, nurse, patient}
Laparotomy	{surgeon, anesthetist, nurse, patient}
Lymphectomy	{surgeon, anesthetist, nurse, patient}

Table 1. Agents involved in tasks from Fig. 2

1.2. Guarded LTS for Process Analysis

A guarded LTS (g-LTS) is a transition system whose transitions are labeled by events and guards. It is a structured form of LTS that reduces state explosion through guard abstractions. The g-LTS formalism provides a convenient milestone on the way from a g-HMSC process model to its corresponding LTS; various types of analysis on g-hMSC process models are performed on g-LTS translations [12].

A *guarded LTS* (g-LTS) is defined by a structure (S, E, VAR, δ, s_0, C_0) where:

- S is a finite set of states.
- E is a set of event labels containing MSC interaction events and events associated with tasks –every task T has two built-in events, denoted by T_{start} and T_{end}, associated with its start and end, respectively.
- VAR is a set of process variables defined over E.
- δ is a guarded transition relation: $\delta \subseteq S \times (L \times GUARD) \times S$, where $GUARD$ is the set of Boolean formulae over VAR, and $L = E \cup \{ \tau \}$ (with τ being a label for non-observable events).
- s_0 is the initial state.
- C_0 is a Boolean formula over fluents and tracking variables; it captures an initial context condition playing the same role as in g-HMSCs.

Fig. 3 shows a g-LTS derived from the g-HMSC in Fig. 2a. The guards there appear between brackets.

Every g-LTS transition is labeled by a guard and by an event. Similarly to g-HMSCs, a g-LTS *guard* is a Boolean formula on fluents and tracking variables. Any g-HMSC can be automatically converted into a g-LTS having the same set of traces. The

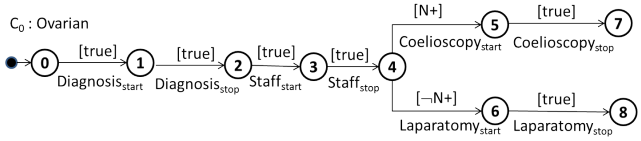

Figure 3. g-LTS derived from the g-hMSC process model in Fig. 2a

rewriting algorithm is detailed in [10, 22]. It extends an algorithm for synthesizing LTS from hMSC [32] so as to cope with guards. The resulting g-LTS abstracts from agents and captures the set of global behaviors covered by the g-HMSC.

The semantics of a g-LTS is defined in terms of guard-free event traces. An algorithm for generating a LTS from a g-LTS can be found in [10, 22].

1.3. Analyzing g-LTS Models

The formal trace semantics of the g-HMSC process language enables a variety of checks on g-HMSC process models converted into g-LTS form.

- *Guard analysis*. The guards on the various alternatives at any decision point in a task flow must be satisfiable in the state reached at that point (for subsequent tasks to be applicable). They may not overlap in that state (for decisions to be deterministic). They must cover all possible cases in that state (no alternative branch is missing).
- *Detection of inadequate decisions*. A decision is said to be *inadequate* if it relies on incorrect information about the state of the process environment. This arises from tracking variables being outdated or inaccurate because of missing tasks or unexpected events. Every decision in a task flow should be adequate.
- *Verification of task preconditions*. All task preconditions should be satisfied along all possible paths of the model.
- *Verification of non-functional requirements on the process*. All timing, resource and cost constraints should be met along every possible path of the process model.

The formal techniques underlying these various types of checks are detailed and amply illustrated in [12]. Each of them is based on a different instantiation of the same generic algorithm for propagating decorations through the process model.

This algorithm propagates generic state decorations through the g-LTS model by symbolic execution until a fixpoint is reached, that is, until no state decoration changes if the propagation is applied once more to every transition. The algorithm accumulates in every state the decorations contributed by every g-LTS execution reaching this state.

As shown in [12], this algorithm is designed to require as little instantiation effort as possible; for each type of check, the user just needs to instantiate the generic decoration together with the rule for propagating decorations through a single state transition. No correctness proofs of the instantiated algorithms are required; the proof of the generic algorithm gets instantiated similarly.

In particular, the decoration algorithm can be instantiated for generating most accurate state invariants [12]. A *state invariant* is an assertion on a specific state of the process that holds every time this state is visited. The *most accurate* state invariant at a state is a Boolean expression on process variables representing the invariant that accounts for all possible process paths reaching that state and only those. In the algorithm instantiation for generating such invariants, the decoration is initially false for every state except the initial state; the latter is decorated with the initial condition C_0. The algorithm propagates invariants through the state machine according to the outcomes of guards and definitions of process variables.

The automation of some of the composition and decomposition operators in the next sections relies on the generation of such invariants to guarantee the desired semantic properties over traces. In the sequel, we will use the following function that maps every g-LTS state to its most accurate state invariant:

$$INV: S \rightarrow \text{INVARIANT}$$

where *INVARIANT* is the set of Boolean formulas over process variables.

For example, the most accurate state invariant generated for state 5 in Fig. 3 is:

$$Ovarian \wedge N+.$$

2. Automating Model Composition and Decomposition Operators

As g-HMSC models have an operational semantics in terms of g-LTS, the automation of our model composition and decomposition operators is based on the intermediate g-LTS formalism too.

The automated application of these operators to g-HMSC models proceeds in three steps.

(a) The input g-hMSC models are transformed into semantically equivalent g-LTS representations according to the algorithm described in [10].
(b) Operator-specific manipulations on the resulting g-LTS are performed so as to meet the semantic properties required by the considered operator. This step includes some basic g-LTS manipulations common to several operators such as event removal, parallel composition, and minimization. These basic operations are described in Section 3.
(c) The resulting g-LTS representation is transformed back into a semantically equivalent g-hMSC. This is the reverse of step (a).

Steps (a) and (c) are the same for all operators. They involve fairly standard compilation/unparsing techniques. This section focusses on the operator-specific step (b).

2.1. The Union Operator

This composition operator generates a process model containing all paths that are valid in the input models and only those. A path is *valid* if all guards are satisfied along it. The output model is restructured by merging all tasks that are equivalent. Roughly, *equivalent* tasks have the same label and have common continuations –see Section 3.3 for a more precise definition.

Fig. 4 shows the result of applying the *Union* operator to the process models for ovarian cancer (Fig. 2 a) and endometrial cancer (Fig. 2 b). Note the new decision node on cancer type appearing there.

The *Union* of two g-LTS models is computed in two steps.

1. A single g-LTS is created from the input g-LTS models. The initial state of this new g-LTS is a decision node whose outgoing guards are the initial context conditions C_0 from each input model, each leading to its corresponding g-LTS.

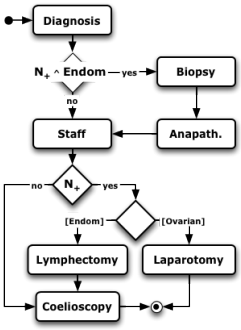

Figure 4. Union of models for ovarian and endometrial cancer treatments

2. The resulting g-LTS is minimized so as to factor out paths common to all input models and integrate their differences in a minimal number of distinguishing paths. As g-LTS have no canonical minimal form, a heuristic algorithm inspired from grammar induction [8, 24] is used for this. This algorithm, described in Section 3.3 below, restructures the g-LTS by merging "equivalent" states while preserving the required set of traces. All guards are then simplified so as to be the weakest for guaranteeing this set of traces. The merge is driven by most accurate invariants generated for each state according to the fixpoint algorithm introduced in Section 1.3 (see Section 3.3 for details).

The initial context condition of the union model is the disjunction of the initial context conditions of the input models.

The *Union* operator is typically applied for model synthesis or for model comparison.

- *Model Synthesis.* Composite process models for large sets of instances are much more difficult to build than dedicated sub-process models involving fewer instances. In our experience, stakeholders tend to explain their procedures by instances. In the medical domain, in particular, they often explain their procedures on a per-pathology basis. As the examples in Fig. 2 and Fig. 4 should suggest, the composite department process model is obtained simply by taking the union over all pathology-centric sub-models (see Fig. 1).
- *Model comparison.* The generated composite model highlights commonalities and differences among the input models.

For the output composite model to be homogeneous and lexically coherent, the input models should be comparable; they should share the same terminology and the same granularity. It may therefore be sometimes necessary to adapt input models before computing their union. The g-HMSC task refinement/aggregation mechanism may be used to enforce the latter assumption.

2.2. The Restriction Operator

This decomposition operator generates a simpler process model containing a subset of valid paths from the input model according to some specific condition called *restriction*

condition. The latter is a Boolean expression on process variables that characterizes a subclass of instances meeting the condition.

For example, we may want to restrict the composite department model in Fig. 4 to highlight the case of ovarian cancers only. The restriction condition is thus *Ovarian*. The restricted model obtained is the one in Fig. 2 a.

As another example, we might want to project the composite department model in Fig. 4 on patients with no lymph node invasion. The restriction condition is thus $\neg N+$. The resulting model in Fig. 5a has one task sequence only. This shows that, when there is no lymph node invasion, the treatment is the same for ovarian and endometrial cancers; no decision node remains that distinguishes between those two cancers.

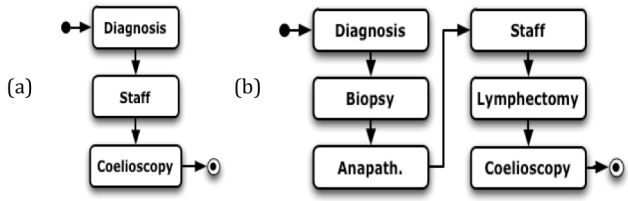

Figure 5. Restriction of the model in Fig 4:
(a) on $\neg N_+$, (b) on $N_+ \wedge$ Endom

Note that the restriction condition on process variables must not necessarily be atomic. Fig. 5b shows the restriction of the model in Fig. 4 according to the condition $N_+ \wedge Endom$.

The *Restriction* of a g-LTS model to a given restriction condition *RC* is computed in three steps, illustrated on the restriction of the g-LTS in Fig. 4 to $RC = \neg N_+$.

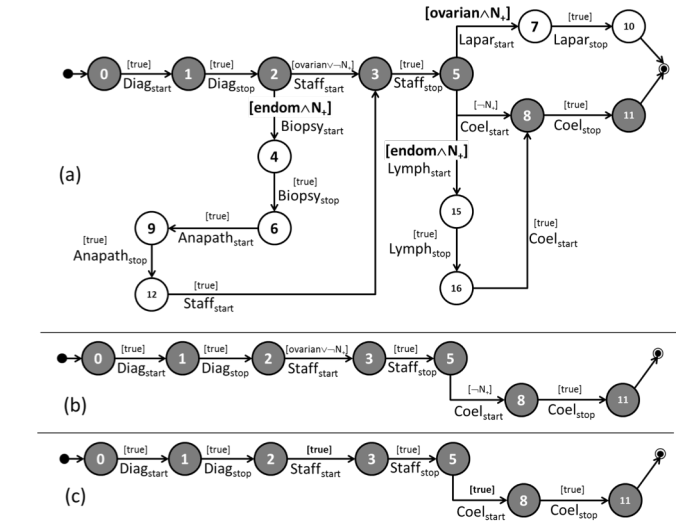

Figure 6. Restriction of the model in Fig 4 on ¬ N+:
(a) after decoration, (b) after removing unreachable states, (c) final g-LTS after guard simplification

1. The input g-LTS is decorated with state invariants using $C_0 \wedge RC$ as invariant at the initial state to start the propagation, where C_0 is the initial context condition of the input model. States becoming unreachable due to the restriction

condition are thereby identified; such states end up with a *false* state invariant. They are shown in white in Fig. 6a.
2. All unreachable states are removed (see Fig. 6b).
3. The resulting g-LTS is minimized in order to obtain a more compact model while preserving the accepted traces (see Fig. 6c). All guards are simplified to their weakest form to ensure this. The counterpart at the g-HMSC level amounts to remove unnecessary decision nodes having a single outcome. For example, the guards from states 2 and 5 in Fig. 6b are removed.

The initial context condition of the restricted model is the conjunction of the initial context condition C_0 of the input model and the restricted condition *RC*. The latter should be stronger than C_0 for the operator to have a restriction effect. Conversely, if *RC* does not satisfy C_0, the output will be an empty model.

The *Restriction* operator is typically applied to help readers understand complex processes or to facilitate their analysis.

- Composite models may capture numerous task sequences followed by process instances having different characteristics. Showing the model for a subset of them may increase its readability.
- Larger erroneous models may result in slower tool feedback that may be harder to understand. Analyses on smaller models of subclasses of process instances may produce faster and more understandable results.

2.3. The Focus Operator

This decomposition operator "projects" a process model on a subset of tasks, called *focus set*. All tasks not included in this set are removed in a semantically consistent way.

For example, Fig. 1 suggests that focusing clinical pathways on a specific department yields pathology-centric department workflows. A more specific example of focused model is shown in Fig 7. The *Focus* operator there yields the patient's view (Fig. 7a), the oncologist's view (Fig. 7b), and the surgeon's view (Fig. 7c) of the department workflow in Fig. 4. The focus set includes all tasks where the corresponding agent is directly involved (see Table 1).

The *Focus* of a g-LTS model on given tasks is computed in three steps.

1. All events not involved in the focus set are replaced by non-monitorable τ events (see Section 3.1.1 hereafter).
2. The latter events are removed using the τ-removal algorithm (see Section 3.1.2 hereafter).
3. The resulting g-LTS is minimized (see Section 3.3).

The *Focus* operator is typically applied for view extraction, model explanation, or for structuring process documentation.

- A simpler model customized to a specific agent view may increase the involvement of the corresponding agent.
- Simplifying a complex model by showing only details relevant to the reader makes the model easier to explain.
- A presentation by agent or department provides a natural organization for complex documents.

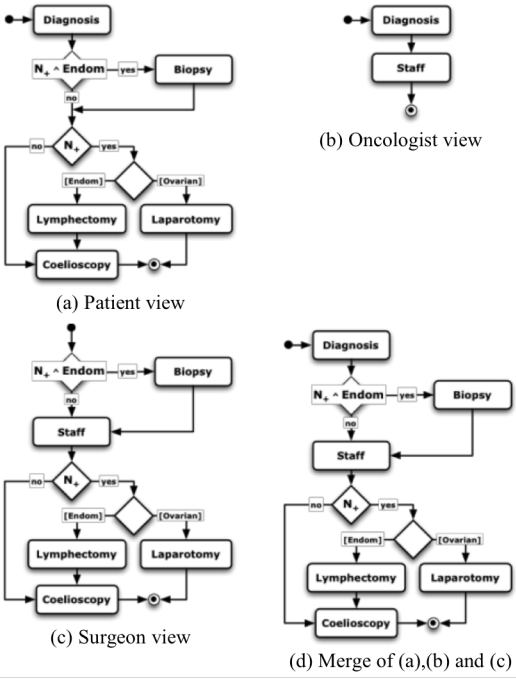

Figure 7. (a-c) Agent views of department workflow, (d) Merge of these agent views

2.4. The Merge Operator

This composition operator is the reverse of *Focus*. It generates a semantically consistent parallel composition of the input process models, where the processes from the input models synchronize on shared tasks.

For example, let us consider the patient's, oncologist's, and surgeon's views of the department workflow in Fig. 7 a-c. Fig. 7 d shows the merge of these three workflows that provides a global view of the department workflow for those three agents.

The *Merge* of g-LTS models is computed in two steps.

1. The input g-LTS models are composed in parallel; all input processes synchronize on shared events but not on guards. This parallel composition of g-LTS is detailed in Section 3.2.
2. The compound g-LTS is minimized (see Section 3.3).

Our *Merge* operator here composes processes by synchronizing them on shared tasks only. This is different from the variant operator in [11] which also uses timing information for defining the composition. In [11], input processes share their timeline; time elapses at the same rate for all of them. The resulting compound model shows which tasks may be applied concurrently and which ones need to be applied sequentially in view of timing constraints.

A new *Merge* operator is introduced here as timing information is in practice not necessarily available. It may thus always be applied even without such information.

The *Merge* operator is typically applied for global analysis, view integration, or for modeling and analysis of parallel workflows.

- Beyond multi-pathology models, the analysis of merged process models allows us to check whether required properties of the input processes still hold at a more global level when they are taken together.
- It is worth merging agent-specific views on a model in order to get a more global view integrating the individual views (e.g., the department view in Fig. 1) or even to obtain the full composite model.
- A process instance may be involved in multiple workflows. For example, a patient may follow multiple clinical treatments –e.g., one for colorectal cancer and another for diabetes. Undesired "feature interactions" or conflicts [23] may occur between the corresponding processes. For example, a task in one input model might change process variables of another input model which might lead to wrong decisions. Analyses on the merged models may help detecting such conflicts –e .g., by model-checking the merge against desired properties [10].

As already noted for the *Union* operator, the input models to be merged should share the same terminology and the same granularity.

3. Base Operations for g-LTS Composition and Decomposition

This section details basic g-LTS manipulations used in some of the steps for model composition/decomposition in the previous section.

3.1. Event Removal From a g-LTS Model

The *Focus* operator was seen to require specific g-LTS events to be removed (see Section 2.3). This proceeds in two steps.

1. The events to be removed are *hidden,* that is, they are replaced by τ-events.
2. These τ-events are removed using the τ-removal algorithm; the latter produces a new g-LTS with equivalent behavior.

3.1.1. g-LTS hiding

The hiding of a set of labels L in g-LTS $G = (S, E, VAR, \delta, s_0, C_0)$ yields the g-LTS

$$G \setminus L = (S, E, VAR, \delta_{hidden}, s_0, C_0),$$

where δ_{hidden} is the smallest relation meeting the following rules:

$$\frac{G \xrightarrow{<g,e>} G'}{G\setminus L \xrightarrow{<g,e>} G'\setminus L}\ e \notin L \qquad \frac{G \xrightarrow{<g,e>} G'}{G\setminus L \xrightarrow{<g,\tau>} G'\setminus L}\ e \in L$$

In these rules, the notation $G \xrightarrow{<g,e>} G'$ means that the g-LTS

$$G = (S, E, VAR, \delta, s_0, C_0)$$

transits into the g-LTS

$$G' = (S, E, VAR, \delta, s_1, C_1)$$

through a guard *g* and an event *e* from its initial state, that is,

- $(s_0, (g\ ,\ e), s_1) \in \delta$,
- $(C_0 \wedge g)_{|e} = C_1$,

where $X_{|e}$ denotes the Boolean expression obtained from X after application of event e. This operation meets the semantics of process variables; their value is defined in terms of event applications along event traces, see [12].

3.1.2. τ-removal

The removal of non-observable transitions works in a way similar to known algorithms on finite automata [18, 26]. In case of g-LTS models, however, guards must be taken into account to preserve the trace semantics.

Removing τ-transitions from a g-LTS

$$G = (S, E, VAR, \delta, s_0, C_0)$$

yields the g-LTS

$$G' = (S, E, VAR, \delta', s_0, C_0)$$

where

- $\delta' = \{ (s, g_\tau \wedge g, e, t) \mid \exists\, u \in S$ such that $(u, g, e, t) \in \delta, e \in E,$ and $(g_\tau, u) \in \tau\text{-CLOSURE}(s)\}$,
- τ-CLOSURE(s) denotes the set of pairs (g, t) such that there is a path from s to t made of τ-labeled transitions only and whose conjunction of guards is g.

3.2. Parallel Composition of g-LTS Models

The *Merge* operator was seen to require a parallel composition of g-LTS models to be computed (see Section 2.4).

Let $G_1 = (S_1, E_1, VAR_1, \delta_1, s_1, C_1)$ and $G_2 = (S_2, E_2, VAR_2, \delta_2, s_2, C_2)$ denote two g-LTS models. Their parallel composition defines the following g-LTS:

$$G_1 \parallel G_2 = (S_1 \times S_2, E_1 \cup E_2, VAR_1 \cup VAR_2, \delta, (s_1, s_2), C_1 \wedge C_2),$$

where δ is the smallest relation satisfying the following rules:

$$\frac{G_1 \xrightarrow{<g,e>} G_1'}{G_1 \| G_2 \xrightarrow{<g,e>} G_1' \| G_2}\ e \notin E_2 \qquad \frac{G_2 \xrightarrow{<g,e>} G_2'}{G_1 \| G_2 \xrightarrow{<g,e>} G_1 \| G_2'}\ e \notin E_1$$

$$\frac{G_1 \xrightarrow{<g_1,e>} G_1', G_2 \xrightarrow{<g_2,e>} G_2'}{G_1 \| G_2 \xrightarrow{<g_1 \wedge g_2,e>} G_1' \| G_2'}\ e \neq \tau$$

The compound g-LTS behaves asynchronously while synchronizing on shared events –but not on guards. In the composition, the guard associated with a shared event is the conjunction of the guards associated with this event in the respective input g-LTS models.

3.3. Minimization of a g-LTS Model

Similarly to other state formalisms such as statecharts [17], g-LTS have no known minimal normal form. Even without any algorithm guaranteed to generate a minimal g-LTS, the composition/decomposition operators in Section 2 should produce process models that are as compact as possible while meeting the semantic properties required on them. A heuristic algorithm is therefore proposed.

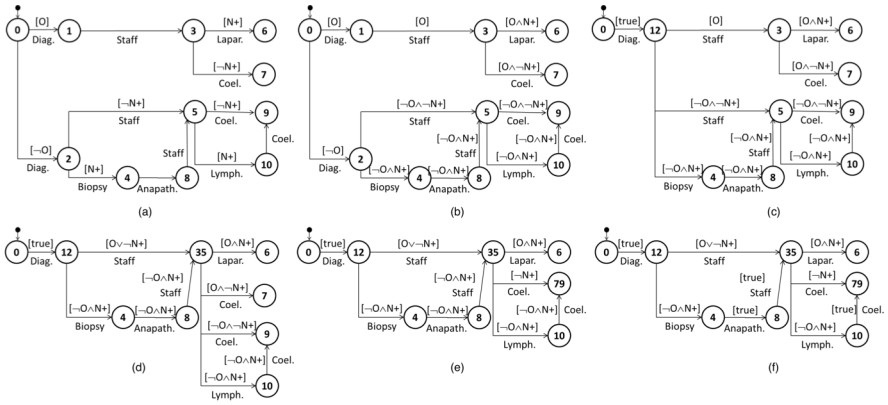

Figure 8. Minimization through state merging

The algorithm is inspired from state-merging algorithms for automaton induction [8, 24]. Roughly, state-merging algorithms reduce the state space of an input automaton by successively merging pairs of equivalent states. *Equivalent* states are those sharing continuations, that is, states sharing at least one outgoing transition labelled with the same event. Merging a state pair (q, q') may require further merging of subsequent state pairs to obtain a deterministic solution; shared continuations of q and q' are folded up by such further merges. At every merge, the algorithm checks desired properties. Here, we need to check that no behavior is added. If the target property is violated, the merging of state pair (q, q') is discarded; the algorithm then continues with the next candidate pair.

This algorithm produces a new g-LTS model meeting the following properties:

- the set of traces of this new g-LTS is exactly the same as the set of traces of the original g-LTS model;
- all guards are simplified so as to be the weakest ones for covering this set of traces;
- the model has fewer states than the original g-LTS.

The minimization algorithm is illustrated in Fig. 8 on the g-LTS model resulting from the union of the two process models in Fig. 2. For lack of space, the event labels are shortened and represent both the *start* and *end* events associated with the corresponding task. The process variable O is *true* for patients having an ovarian cancer and *false* for those having endometrial cancer. The g-LTS model to be minimized is shown in Fig. 8a. The algorithm proceeds in three steps.

Step 1: Guard strengthening with generated state invariant. This step aims at getting all the information captured by most accurate state invariants at guarded transitions before merging. For this,

- the most accurate state invariant is computed for each state using the decoration propagation algorithm introduced in Section 1.3;
- each guard is strengthened by adding the computed most accurate invariant at the corresponding state as conjunct.

For a given g-LTS $G = (S, E, VAR, \delta, s_0, C_0)$, this step produces the new g-LTS $G' = (S, E, VAR, \delta', s_0, C_0)$ where

$$\delta' = \{(s, g \wedge INV(s), e, s') \mid (s, g, e, s') \in \delta\}.$$

Table 2 shows the most accurate invariants generated for each state of the g-LTS in Fig. 8a. The g-LTS after guard strengthening is given in Fig. 8b.

state	Invariant
0	true
1	O
2	¬O
3	O
4	¬O ∧ N+
5	¬O
6	O ∧ N+
7	O ∧ ¬N+
8	¬O ∧ N+
9	¬O
10	¬O ∧ N+

Table 2. Most accurate invariants for g-LTS states in Fig. 8a

Step 2: Invariant-based state merging. Well-selected state pairs are merged using an algorithm inspired from [9]. The pairs being selected are those including states that:

- share similar continuations or are successors of non-deterministic states,
- are non-overlapping.

Two states are said to be *non-overlapping* if:

$$INV\ (state1) \wedge INV\ (state2) = false.$$

The non-overlapping condition is required to ensure that no behavior is added in the minimization process, as explained below.

Consider the g-LTS model in Fig. 8b. *State 0* is non-deterministic on events; the two transitions labelled by event *Diag* lead to merging *State 1* and *State 2*. When merging transitions with the same label, the disjunction of their respective guards is taken as new guard. Here we get:

$$O \vee \neg O = true$$

The merging of *State 1* and *State 2* results in non-determinism on event *Staff* (Fig. 8c). This calls for further merging of *State 3* and *State 5* (Fig. 8d). The process continues while such non-determinism propagates (see Fig. 8e).

As stated before, we need to merge non-overlapping states only. The intuition behind this is suggested in Fig. 9. Let us consider the left fragment in Fig. 9 where two states *s1* and *s2* have a single outgoing transition labelled by (guard *g1*, event *e1*) and (guard *g2*, event *e2*), respectively. As we merge them (see right fragment in Fig. 9), a new trace will be accepted if:

$$INV(s2) \models (INV(s1) \wedge g1)$$

or

$$INV(s1) \models (INV(s2) \wedge g2).$$

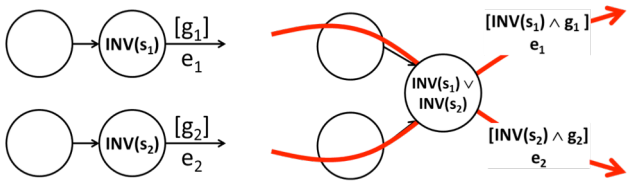

Figure 9. Merging non-overlapping states only

As the weakest possible guard is *true*, a new trace might be introduced if INV(s1) |= INV(s2) or vice-versa. To avoid introducing such prohibited new traces, we merge non-overlapping states only. In Fig. 8a, *State 2* and *State 8* share the same outgoing *Staff* event; they may however not merge as

$$INV(State2) = \neg O$$

$$INV(State8) = \neg O \wedge N_{+}.$$

Merging those two states would add new behaviors indeed; for example, after merge, a patient meeting the initial condition

$$\neg O \wedge N_{+}$$

might loop into the *Biopsy* and *Anapath* events.

Step 3: Guard simplification. All guards resulting from the first two steps are simplified so as to be the weakest for preserving the set of accepted behaviors. This step is the reverse of the first step for guard strengthening; it removes the contextual invariant from each guard. Fig. 8f shows the minimized g-LTS model after guard simplification.

Simplifying the guards of a g-LTS $G = (S, E, VAR, \delta, s_0, C_0)$ yields a new g-LTS $G = (S, E, VAR, \delta', s_0, C_0)$ where

$$\delta' = \{(s, Simplify(g, INV(s)), e, s') \mid (s, g, e, s') \in \delta\}.$$

The simplification operation *Simplify (g, C)* of guard g by context C yields the weakest Boolean expression such that:

$$Simplify\ (g, C) \wedge C = g \wedge C.$$

For example,

$$Simplify\ (P \wedge Q, Q) = P.$$

This step is implemented using standard propositional operators available in Binary Decision Diagrams libraries.

4. Macro-Operators for Multi-Step Model Construction

The model composition/decomposition operators in Section 2 take one or more g-HMSC models as input to produce a single g-HMSC model as output. These operators are thus closed over g-HMSCs and form a basis for a workflow algebra. In the AI planning tradition, *macro-operators* can therefore be defined by combining those base operators. This proves useful for more sophisticated model manipulations.

In the context of Fig.1, typical macro-operator applications would include the following operator combinations.

- To produce a specific hospital department workflow from multiple clinical pathways, we first apply the *Union* operator and then the *Focus* operator. For example, the radiotherapy department workflow model is obtained by taking the *Union* of clinical pathway models over all different types of cancer and then the *Focus* on the resulting model with all radiotherapy tasks in the focus set.
- The generation of personal treatment documentations is obtained by applying the *Restriction* operator to get the patient's own treatment and then the *Focus* operator to compute her specific view.

5. Tool Support

The model composition/decomposition operators in Section 2 are implemented and integrated in our earlier toolset for process model analysis [12]. The toolset's g-LTS level is implemented as an automaton library; all g-LTS composition/decomposition operators are implemented there as a service layer to the g-HMSC level. Binary Decision Diagrams [1] are used for capturing and manipulating g-LTS guards efficiently thanks to the Cudd library [31] also used in the NuSMV model checker [6].

Our toolset makes use of a declarative guarded command language for users to enter structured process models. The g-HMSC models expressed textually in this input language are instantly visualized in the graphical syntax used in the paper while being compiled into an equivalent g-LTS representation. The latter is manipulated by our various checkers [12] and by the composition/decomposition operators discussed in this paper. The resulting g-LTS representations (and error messages) are then converted back to g-HMSCs by use of decompilation techniques [5]. A g-LTS is seen as a control flow graph from which *goto* constructs are incrementally replaced by higher-level constructs such as *if-then-else* statements, *while* loops, and so forth.

Currently, the *Union*, *Restriction* and *Focus* operators are fully supported in roundtrip mode. The *Merge* operator requires some further work as g-LTS models are not all restructurable in our current input language; the latter should first be extended with an explicit *fork/join* construct with corresponding decompilation algorithms.

6. Evaluation

Our model composition/decomposition operators were applied in the context of a new particle-therapy center project envisioned by a key industrial player. Particle therapy is a form of radiotherapy that uses beams of energetic protons, neutrons, or positive ions for treating a wide variety of cancers.

Two input documents were available that separately described 18 pathology-centric treatment processes. Each of these 18 process descriptions consisted of a linear scenario showing a specific sequence of moves through the center with corresponding pathology-specific treatment in dedicated rooms containing the required equipment. These 18 scenarios were thus structured according to the complex topology of the center. No global composite model integrating those 18 patient routes was available; some project partners tried to build one but did not succeed because of the complexity

of the overall composite process. The global workflow was thus missing for the projected center.

Our goal was therefore to build an overall composite model for this global workflow from those 18 scenarios by use of our operators (regardless of the appropriateness of the provided input scenarios).

Fig 10 gives simplified and anonymized excerpts from the documents available to us.

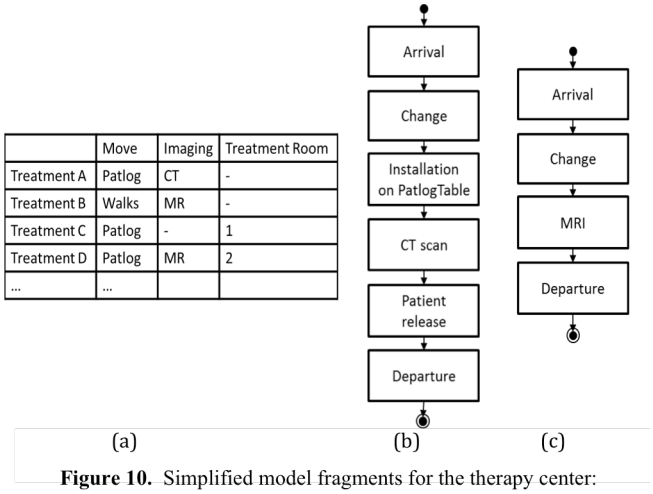

Figure 10. Simplified model fragments for the therapy center: (a) table for extracting variables, (b) workflow for treatment A, (c) workflow for treatment B

The first document is summarized in a table describing the main characteristics of all treatments given in the center. A simplified version of this table is outlined in Fig. 10a.

- In treatment A, for example, patient X only undergoes imagery through a CT scan (computed tomography). She is transported to the imagery room using a treatment couch trolley (referred as "patlog" in Fig. 10). She does not undergo radiotherapy.
- In treatment B, patient Y does not use a trolley. He undergoes MR imaging (Magnetic Resonance) without radiotherapy treatment.

Such a table allowed us to identify process variables such as "move_by_patlog" which is *true* if the patient is transported by patlog trolley in the center and *false* otherwise. The full table allowed us to extract 10 process variables.

The second document described the task sequence for each treatment. The 18 decision-free workflows involve medical tasks (imagery, radiation, etc.) as well as administrative and logistics tasks performed by employees. Fig. 10b and Fig. 10c provide simplified examples for treatments A and B. The initial context condition for these simple workflows was easily derived using available tables such as the one in Fig. 10 a.

Our challenge was to produce a composite workflow model for the whole center that covers all 18 treatment scenarios. First, we tried to build the composite model manually. Like previous attempts by others, we did not succeed. Building a single

accurate workflow model integrating all 18 treatments, that are interfering at places, turned to be a daunting, error-prone task.

Using our tool, the *Union* of the 18 workflows was computed to generate the entire composite model. To get a rough idea of what the resulting model looks like, a portion of it is outlined in Fig. 11.

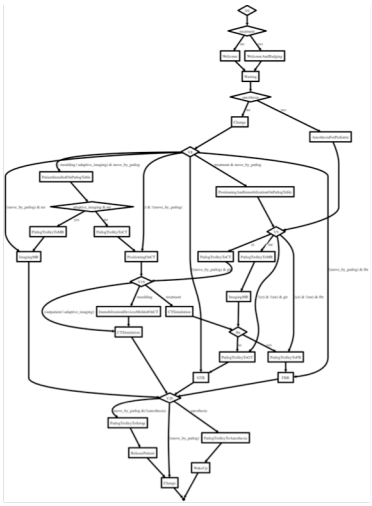

Figure 11. Generated model fragment for union of all treatments in the center (outline)

We also wanted to validate this model with domain experts. The model turned out to be too big for careful manual analysis. We therefore used our tool to support the validation process. Other operators such as *Focus* and *Restrict* were applied to present various views on the composite model; these views were smaller and much easier to validate. For example, we showed workflows with administrative tasks only (using *Focus*), with medical tasks only (using *Focus* again), and with treatments involving anesthesia (using *Restriction*). Errors were detected on those various views, allowing us to highlight errors in the input task sequences provided to us. After fixing those errors in the input scenarios, a new *Union* workflow was generated again in a few seconds.

Taken together, the 18 task sequences contained 147 task nodes. Their *Union* drastically reduced the number of task nodes to 26 while introducing 9 decision nodes on the 10 identified process variables. These decision nodes prevented new, undesirable behaviors in the model. The task merging produced by the *Union* operator was evaluated as very good by domain experts.

7. Related Work

A conceptual framework for classifying and comparing approaches to model merging is proposed in [2]. This framework can be applied to any type of model. Generic operators are suggested there for merging two models into one, for slicing a model to produce a partial view, and so forth. These operators are expected to meet different algebraic properties such as associativity and commutativity. This framework might be used to classify our operators. For example, *Merge* and *Union* somewhat instantiate the

"merge" generic operator in [2] whereas *Restriction* and *Focus* instantiate the "slice" generic operator. Our four operators in addition meet the properties of idempotency, commutativity, associativity and totality.

Another instantiation of the "merge" operator from [2] is found in [28] for composing hierarchical state machines. It appears related to our *Union* operator with notable differences, however.

- Our *Union* operator requires process operands to have the same granularity. This limitation is overcome in [28] through a *Match* operator aimed at finding relationships between the input models before computing their union. The latter operator is heuristic and uses both static and behavioral properties to match state pairs in the input models. User intervention may be required to find correspondences between the input models. Such *Match* operator might nicely complement our framework.
- Like our *Union* operator, the operator in [28] produces a combined model in which variant behaviors of the input models are parameterized using guards on their transitions. In our approach, g-HMSC task occurrences may change the value of process variables. Our guards are added to preserve the required set of traces; they depend on state invariants. Our decoration algorithm must therefore be used in the implementation of the *Union* operator to compute invariants and guards.

Workflow views are also considered in workflow management systems and web services [4, 14, 15, 25]. They are used for composing services while keeping private parts of the underlying process hidden. Such views are obtained by application of so-called *Abstraction* or *Hiding* mechanisms. The latter somewhat correspond to our *Focus* operator. Mechanisms corresponding to our *Union*, *Restriction* and *Merge* operators are apparently not considered in this literature.

Formal operators are also available on Petri nets [29]. Inspired from relational algebra, they are not restricted to simple *Abstraction/Hiding* mechanisms. They are, however, introduced at the fairly low level of Petri nets –rather than at the level of a process modeling language such as YAWL [33]. No distinction is made between the modeller's level and the level at which the trace semantics of the modeling language is defined. The application of such operators in workflow management situations remains unclear.

The algorithm for computing the union of two g-LTS is inspired from our previous work on synthesizing behavior models from scenarios under constraints [8, 9, 21]. There are two main differences. First, the *Union* operator does not create new traces whereas the induction-driven technique in [8] aims at generalizing scenario behaviors. State invariants are used here to avoid merging states that would result in accepting new traces. Secondly, the *Union* operator has to deal with guards by recomputing them after each merge through guard disjunction. Our previous work makes use of pure event-based LTS.

8. Conclusion

Processes in human-intensive systems may be quite complex, covering multiple views refering to different concerns, agents, resources, locations, non-functional requirements, and so forth. Different views on a composite process may be captured through different

process models. Each of these proves useful for specific purposes including their analysis, documentation, and enactment. Constructing all relevant models manually, in a semantically consistent way, appears very difficult in practice.

The formal techniques in this paper for automated model composition and decomposition address this problem. The *Union* operator allows us to synthesize a composite model from simpler ones. The commonalities and differences among different related workflows may thereby be highlighted through a workflow family model. The *Merge* operator allows us to integrate multiple agent-specific views or to analyze concurrent process views. The *Restriction* operator allows us to filter a composite model on a specific restriction condition in order to facilitate understanding and analysis. The *Focus* operator allows us to filter a composite model on a specific subset of tasks in order to extract agent-specific views or to produce customized documentation.

These composition/decomposition operators, informally introduced in [11], are fully automated here through a variety of manipulations of g-LTS models. They are integrated in our earlier toolset for building and analyzing critical process models [12]. Our experience of model synthesis and view extraction for a real, challenging workflow in a particle-therapy center appears quite promising. Our tool was able to compute complex models by operator applications in a few seconds. It allows on-the-fly model building upon request by domain experts.

Future work should include the elaboration of guidelines for making effective use of our operators together with further research on additional operators and useful macro-operators –in particular, for detaching unfrequent decision paths as explicit exceptions and vice-versa.

Acknowledgment

We wish to thank Antoine Cailliau for helping us in the automated simplification of guards in the g-LTS minimization algorithm. Thanks are also due to François Roucoux and Damien Bertrand for providing interesting examples and material for the case study in Section 6. This work was supported by the Regional Government of Wallonia (PIPAS project Nr. 1017087).

References

[1] S.B. Akers, "Binary decision diagrams", *IEEE Transactions on Computers*, 27(6), 1978, 509-516.

[2] G. Brunet, M. Chechik, S. Easterbrook, S. Nejati, N. Niu, and M. Sabetzadeh, "A *manifesto* for model merging", *Proc. GaMMa '06: Intl. Workshop on Global Integrated Model Management*, 2006, 5-12.

[3] 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.

[4] D. Chiu et al., "Workflow view driven cross-organizational interoperability in a web service environment", *Information Technology and Management* 5(3), 2004, 221-250.

[5] C. Cifuentes, "Structuring decompiled graphs", *Compiler Construction*, Springer-Verlag, 1996.

[6] A. Cimatti, E. Clarke, F. Giunchiglia and M. Roveri, "NuSMV: A new symbolic model verifier", *Proc. CAV'99*, LNCS 1633, Springer-Verlag, 1999.

[7] L.A. Clarke, L.J. Osterweil, and G.S. Avrunin, "Supporting human-intensive systems", *Proc. FoSER'10: FSE/SDP Workshop on Future of Software Engineering Research*, ACM, 2010, 87-92.

[8] C. Damas, B. Lambeau, P. Dupont, and A. van Lamsweerde, “Generating annotated behavior models from end-user scenarios”, *IEEE Trans. on Software Engineering*, 31(12), Dec. 2005, 1056-1073.

[9] C. Damas, B. Lambeau, and A. van Lamsweerde, “Scenarios, Goals, and State Machines: A Win-Win Partnership for Model Synthesis”, *Proc. FSE’06: 14th ACM Intl. Symp. on the Foundations of Software Engineering*, Portland (OR), November 2006.

[10] 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.

[11] C. Damas, B. Lambeau, A. van Lamsweerde, "Transformation operators for easier engineering of medical process models”, Proc. SEHC’2013: *5th Intl. Workshop on Software Engineering in Health Care (SEHC)*, San Francisco, ACM-IEEE, May 2013, 39-45.

[12] C. Damas, B. Lambeau, A. van Lamsweerde, “Analyzing Critical Decision-Based Processes”, *IEEE Transaction on Software Engineering* Vol. 40 No. 4, April 2014, 338-365.

[13] M. Dumas, W. van der Aalst, A. ter Hofstede, *Process-Aware Information Systems*. Wiley, 2005

[14] J. Eder and A. Tahamtan, “Temporal consistency of view-based interorganizational workflows”, *Information Systems and e-Business Technologies*, 2008, 96-107.

[15] R. Eshuis and P. Grefen, “Constructing customized process views”, *Data & Knowledge Engineering*, 64(2), 2008, 419-438.

[16] D. Giannakopoulou, J. Magee, “Fluent model checking for event-based systems”, *Proc. ESEC/FSE 2003*, Helsinki, 2003.

[17] D. Harel, "Statecharts: A visual formalism for complex systems", *Science of computer programming* 8(3), 1987, 231-274.

[18] J. E. Hopcroft and J. D. Ullman, *Introduction to Authomata Theory, Languages and Computation*, Addison-Wesley, 1979.

[19] ITU, *Message Sequence Charts*. Recommendation Z.120, Intl. Telecom Union, Telecom. Standardization Sector, 1996.

[20] 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, pp. 214-224, 2009.

[21] B. Lambeau, “State-merging DFA induction algorithms with mandatory merge constraints”, *Proc. ICGI08: 9th International Conference on Grammatical Inference*, 2008, 139-153.

[22] B. Lambeau, *Synthesizing Multi-Model Views of Software Systems*, Ph.D. Thesis, Université catholique de Louvain, 2011.

[23] A. van Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, 2009.

[24] K. Lang, B. Pearlmutter and R. Price, “Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm”, in *Grammatical Inference*, LNAI 1433, Springer-Verlag, 1998, 1-12.

[25] D. R. Liu and M. Shen, “Workflow modeling for virtual processes: an order-preserving process-view approach”, *Information Systems*, 28(6), 2003, 505-532.

[26] J. Magee and J. Kramer, *Concurrency: State Models and Java Programs*. Second Edition, John Wiley & Sons, 2006.

[27] R. Milner, *Communication and Concurrency*. Prentice-Hall, 1989.

[28] S. Nejati, M. Sabetzadeh, M. Chechik, S. Easterbrook, and P. Zave, “Matching and merging of variant feature specifications”, *IEEE Transactions on Software Engineering*, *38*(6), 2012, 1355-1375.

[29] V. Pankratius and S. Wolffried, “A formal foundation for workflow composition, workflow view definition, and workflow normalization based on Petri nets”, *Proc. 2nd Asia-Pacific Conf. on Conceptual Modeling*, 2005.

[30] M. Renholm, H. Leino-Kilpi, T. Suominen, “Clinical pathways: a systematic review”, *Journal of Nursing Administration* 32(4), 2002,196-202.

[31] Somenzi, Fabio. "CUDD: CU decision diagram package release 2.3", University of Colorado at Boulder, 1998.

[32] S. Uchitel, J. Kramer, and J. Magee, “Synthesis of Behavioral Models from Scenarios”, *IEEE Trans. Softw. Engineering*, 29(2), 2003, 99-115.

[33] W. van der Aalst et al., “YAWL: yet another workflw language”, *Information Systems* 30(4), 2005.