Verification of Timed Healthcare Workflows Using Component Timed-Arc Petri Nets

Cristiano Bertolini[1,2], Zhiming Liu[2], and Jiří Srba[3]

[1] Federal University of Pernambuco, Brazil
[2] UNU-IIST, Macau
[3] Aalborg University, Denmark

Abstract. Workflows in modern healthcare systems are becoming increasingly complex and their execution involves concurrency and sharing of resources. The definition, analysis and management of collaborative healthcare workflows requires abstract model notations with a precisely defined semantics and a support for compositional reasoning. We use the formalism of component-based timed-arc Petri Nets (CTAPN) for modular modelling of collaborative healthcare workflows and demonstrate how the model checker TAPAAL supports the verification of their functional and non-functional requirements. To this end, we use CTAPN to define the semantics of the healthcare domain specific graphical notation Little-JIL, extended with timing constrains, and apply it to the case study of blood transfusion. The value added in general, and to Little-JIL in particular, is the formal support for modelling, analysis and verification with the explicit treatment of the timing aspects.

1 Introduction

It is now a global quest to solve the pressing problems of the constantly growing demand with limited resources in providing people with safer, more effective, more patient centered, and more timely, efficient and equitable health systems. The advances in computing and communication technologies provide the potential for solutions by developing integrated health information systems (IHIS) aimed at providing effective support to secure sharing of information and resources across different healthcare settings and collaborative healthcare workflows among different care providers. However, the workflows to provide healthcare within an integrated system become more complex than the traditional sequential processes with standalone systems. Their execution involves concurrency and sharing of resources through synchronization and interaction. They are obviously safety critical, with several non-functional performance prerequisites including timing requirements on top of the functional requirements. Workflow definitions, analysis and management need abstract model notations that have a precisely defined semantics and we need to develop techniques for compositional design and verification in order to ensure their correctness.

Due to the business models and practice of health organizations and healthcare professionals, hospitals and doctors in particular, a rigorous validation and

J. Weber and I. Perseil (Eds.): FHIES 2012, LNCS 7789, pp. 19–36, 2013.

automation of the healthcare processes and workflows are often lacking behind. While formal modelling, validation and automation of general workflows have been an active area of research (see e.g. [6, 22, 25–28]), there has been less work done in the area for healthcare workflows [4,14,18,19,21]. In particular, there is so far only limited effort in the development of healthcare domain specific modelling notations with tool support for workflows. Such a generally accepted notation would form an important step towards the application of formal techniques and tools for modelling and validation in the area of healthcare. According to our knowledge, one of the few established healthcare-domain modelling notations is Little-JIL [29]. It is based on a graphical notation jointly developed by experts in software engineering and healthcare professionals.

There is no formal abstract semantics defined for Little-JIL. Instead, a compiler is developed to translate a Little-JIL model into a finite-state machine (FSM). Simulation of a workflow is done by executing the finite state machine. and properties of a workflow are specified as a property of the state machine, instead of its direct formulation in Little-JIL notation, and FSM-based model checkers can be used for the verification of requirements. The main drawback of Little-JIL semantics (via its associated FMS formalism) is the lack of hierarchy, thus there is no support for modular (compositional) modelling and verification. Furthermore, there is only a limited support for timing in Little-JIL (expressed via durations) and the timing aspects are not reflected in the FSM semantics.

We propose a new semantical approach for Little-JIL workflows based on *Component Timed-Arc Petri Nets* (CTAPNs), a component-based version of Petri nets where timing information is attached to tokens. A CTAPN supports modular specification and verification in general and has an efficient model checker called TAPAAL [9] implementing all the necessary modelling features. In order to demonstrate the suitability of CTAPN for modelling healthcare workflows, we relate it to Little-JIL by showing how to translate the Little-JIL flow primitives into CTAPNs. Our main focus is on the representation of non-functional requirements, mainly the timing; we deliberately stay on a semi-formal level in our translations instead of the fully formal technical treatment that has been already developed for untimed workflows (see e.g. [24] for an overview). We believe that our presentation style will help to highlight the intuition behind the translation, focusing mainly on the timing aspects.

The translation that we present reflects the graphical similarity of CTAPN and Little-JIL notations. CTAPN is a natural choice of our modelling notation also because Petri nets are among the most popular models of workflows in general [11] and because of the available tool support. Compared to the traditional Petri net models of workflows, CTAPNs support a simple and intuitive representation of continuous timing, that is yet expressive enough for describing advanced timing constraints used in workflows. We use the workflow of blood transfusion [8], the benchmarking case study of Little-JIL extended with timing, to illustrate the applicability of our approach.

Related Work. Modelling of workflows using workflow nets is a classical topic (see e.g. [24–27]), however, as timing is becoming a safety critical aspect of healthcare

workflows (e.g. blood unit expiration time), extensions of the existing approaches should be studied. For example in [10, 17] the authors study time constrained workflow modelling in the formalism of Time Petri Nets (TPN) [20], a different model than CTAPNs. However, some time requirements like global deadlines, a feature that can be easily modelled in CTAPNs, have to be precomputed [10] due to the missing modelling primitives in the TPN model. Other approaches [19,21] translate timed YAWL workflows into untimed model checkers via the explicit-time method (clocks are encoded as integers). As advocated by Lamport [16], such methods can compete with real-time model checkers as long as the constants used in the models are small. However, this is not always the case for healthcare workflows—in our case study we used constants of sizes up to 90 and deadlines above 200 minutes with the average size of nonzero constants being 28 (for the model with single patient). The advantage of the CTAPN model and its model checker TAPAAL is that they support real-time verification using the data structure DBM that is less sensitive to the sizes of the constants [16].

Our earlier work [4] proposes to apply the rCOS model-driven method [7, 15] for modelling healthcare workflows, including the blood transfusion case study modelled in CSP. No analysis techniques or tool support are studied in [4]. In the present paper we take the framework one step forward and focus on the modelling of the process view of the workflow with CTAPN and assess the applicability of the verification techniques for CTAPN. Another related work [18, 23] presents a different formal approach for modelling and verification of workflows, including case studies in healthcare. The method and tool is called NOVA. It supports graphical modelling and implements a translation to DiVinE model checker, including timing aspects in the form of delays and durations. The approach considers only discrete time (indirectly simulated in DiVinE) and as remarked in [18], the required verification time is often unacceptable. State-space reduction techniques that include timing aspects are currently under investigation.

Finally, we relate our modelling approach to the domain specific language Little-JIL that already contains a translation into finite-state machines, however, still without the possibility to verify timing aspects. Citing [8]: "Properties B.9 and B.11 involve real time (e.g. an event needs to happen 15 minutes after another event). The current version of FLAVERS and PROPEL does not support real time properties and, thus, we were not able to specify B.9 and B.11 in PROPEL nor check them with FLAVERS." To the best of our knowledge, there is no further published work on tool-supported verification of Little-JIL timing aspects.

2 Blood Transfusion Case Study and Little-JIL

This section gives an informal introduction to the graphical modelling notation of Little-JIL. We will use the blood transfusion workflow for illustration and we start by introducing this case study.

2.1 Blood Transfusion Case Study

We consider the blood transfusion case study from the Little-JIL benchmark [1]. This medical workflow involves a *nurse*, a *doctor*, a *blood bank* and a *patient*; we call them the resources [4]. The patient is required to provide his/her personal details to the nurse. The nurse then carries out the transfusion procedure: (1) the nurse checks the patient's consent with the transfusion; (2) the nurse waits for a doctor to complete the order; (3) the nurse checks the patient's blood type and availability of the blood type in the blood bank; at the same time she books the transfusion room; (4) the blood product is picked up and the transfusion starts; (5) the nurse monitors every 15 minutes the patient and checks for any reaction; (5a) if a reaction occurs, the nurse can try to adjust the IV access; (5b) if there is still a reaction, the nurse must immediately stop the transfusion and informs the doctor; (6) when the transfusion is finished, the transfusion room is released and sterilized and (7) the nurse checks out the patient.

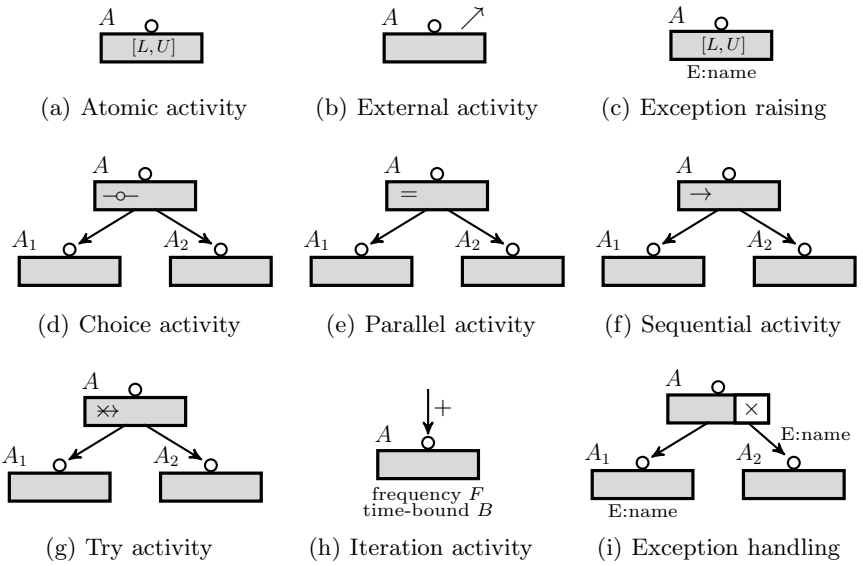
These are only the main steps of the workflow; compensations, exception handling and other details including the timing intervals for all tasks are described in the Little-JIL notation that we shall now introduce.

2.2 Little-JIL

Little-JIL [29] is a visual language used to describe the order and communication between its steps with a particular focus on healthcare workflows. The basic building constructs of Little-JIL are shown in Figure 1. The small circles represent the interface of each activity together with its name and the subactivities are to be interpreted from left to right. In this short overview we represent all Little-JIL primitives in their binary form only[1] and we focus mainly on the flow primitives. Full treatment of the syntax can be found in Little-JIL 1.5 language report [29]. We equip the Little-JIL constructs with more refined timing aspects so that the activities have an execution interval, expressing the uncertainty in the exact duration of the activity, and constructs like iteration allow for time-guarded executions.

A Little-JIL model consists of a finite set of rooted diagrams. Each diagram is either an *atomic activity* (see Figure 1(a)) together with a time interval *[L,U]* where L is the shortest and U the longest execution time of the activity. Activities can be also exported (calling subactivities specified in separate Little-JIL diagrams) as indicated by the arrow on the external activity presented in Figure 1(b). An activity step can also raise exceptions. In this case a label *E:* <*exception name*> is displayed below the step box like in Figure 1(c). If the present activity cannot be successfully finished, an exception is raised, the control flow is interrupted, and the exception is passed to the corresponding exception handler. The remaining flow primitives are as follows. Figure 1(d) shows the *choice* constructor. When the choice is activated only one of the sub-steps is executed. Figure 1(e) shows the *parallel* constructor of two steps. In this case all

[1] The syntax can be in a straightforward manner extended to multiple subactivites.



(a) Atomic activity (b) External activity (c) Exception raising

(d) Choice activity (e) Parallel activity (f) Sequential activity

(g) Try activity (h) Iteration activity (i) Exception handling

Fig. 1. Little-JIL workflow primitives

sub-steps are executed concurrently and the parallel activity terminates as soon as all its subactivities terminated. Figure 1(f) shows the *sequential* constructor of two steps that are executed sequentially from left to right. Figure 1(g) shows the *try* constructor that allows to try the sub-steps from left to right until one of them succeeds and then the try activity terminates too. Figure 1(h) shows the *iteration* constructor where the activity A is repeatedly executed every F time units until its overall duration reaches the bound B. Then the iteration activity terminates. Figure 1(i) shows the *exception* constructor. The exception handler is also a sub-step but it is specified on the right of the step bar and its scope is for all subactivities, including the exported ones. This is an important feature of Little-JIL since workflows in general handle many exceptions.

3 Modelling of Little-JIL Workflow in TAPN

We shall now introduce component timed-arc Petri nets (CTAPN) and present a compositional translation of Little-JIL constructs into the timed nets.

3.1 Introduction to Component Timed-Arc Petri Nets

Petri nets are a graphical formalism for conceptual modelling of distributed systems. We use a particular real-time extension of Petri nets called Timed-Arc Petri Nets (TAPN) [5,12] where an age (nonnegative real number) is associated to every token in the net and input arcs carry time intervals that restrict the ages of tokens suitable for transition firing.

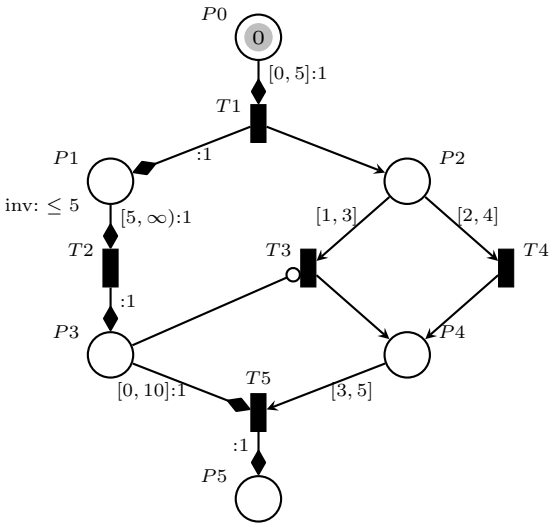


Fig. 2. A TAPN Model of a Simple Workflow

We shall first informally introduce the model. A fully formal treatment of this model can be found e.g. in [13]. Figure 2 shows a simple TAPN model of a workflow process. The net consists of six *places* drawn as circles and five *transitions* drawn as rectangles. The arrows depict different types of *arcs* that connect either places to transitions (input arcs) or transitions to places (output arcs). The dynamics of the net is described by *markings*, i.e. distributions of tokens, each with its own time-stamp (age), in the places of the net. In our example there is one token of age 0 in place $P0$. If tokens of suitable ages are present in all places connected by input arcs to a given transition, the transition gets *enabled* and it can *fire* with the effect of consuming one token of appropriate age from every input place and producing tokens to all places connected with the transition via output arcs. Alternatively, the net can perform a *delay* where all tokens in the net grow older by a given time delay (real number).

In the extended TAPN model we can identify three types of arcs: normal arcs, transport arcs and inhibitor arcs. *Normal arcs* are drawn using a simple arrow tip. Moreover, normal arcs from places to transitions, like the one from place $P2$ to transition $T4$, carry time intervals restricting the ages of tokens that can be consumed by these arcs. Output arcs do not have any associated time interval as the newly produced tokens are by default of age 0. A pair of transitions with diamond-shaped arrow tips, like e.g. from $P0$ to $T1$ and further to $P1$, represent *transport arcs* where the symbol :1 indicates the pairing of input and output transport arcs (in principle there may be several pairs of transport arcs associated with the same transition). The intuition is that once a token is moved along a pair of transport arcs, its age is preserved and not reset. As the whole left-side path from $P0$ to $P5$ consists of only transport arcs, this allows us to measure the total running time of the net since its initialization. Finally, the arc with a circle tip between $P3$ and $T3$ represents an *inhibitor arc*. A presence of

at least one token in $P3$ disables the firing of the transition $T3$ but if the place $P3$ is empty then $T3$ is enabled (provided that $P2$ has at least one token of age between 1 and 3) and then the inhibitor arc has no effect on its firing. Notice also that the place $P1$ contains the *age invariant* ≤ 5, meaning that only tokens of age at most 5 are allowed in this place. If there is at least one token of age 5 in $P1$ then no further delay transitions are possible and the net is forced to fire some of its currently enabled transitions.

The workflow net can be executed for example as follows. The first task represented by the transition $T1$ can be performed within the first five time units. If its deadline is missed (which is a valid behaviour of the net) then only time delay transitions are possible and the token in place $P0$ is then called *dead*. Assume that the first task is executed at say 4.5 time units. Then the token of age 4.5 is moved from $P0$ to $P1$ and a new token of age 0 is produced into $P2$. Clearly, none of the transitions is enabled but if we wait 0.5 time units (the maximum allowed time delay due to the age invariant in $P1$) then the transition $T2$ can fire (simulating the execution of the second task) and move the token of age 5 from $P1$ to $P3$. In this particular scenario, the age of the token in $P2$ is now 0.5 and hence no further transitions are currently enabled. However, after say 1.5 time units, $T4$ can fire, leaving us with one token of age 0 in $P4$ and one token of age 6.5 in $P3$. Note that the workflow has in principle a choice between executing the third or fourth task (represented by transitions $T3$ and $T4$), however, in this concrete execution $T3$ is disabled due to the presence of a token in $P3$. After the delay of another 3 time units, the last task represented by $T5$ can be finally executed, producing a token of age 9.5 into the final place $P5$, and the workflow successfully terminates.

During the modelling of larger systems it is often the case that the net becomes too large to provide an effective overview of the structure of the model. In order to overcome this problem, we consider a simple component-based extension of the model. Here we divide the design into a number of smaller components (essentially workflow patterns) with a clearly defined interface in terms of *shared places and transitions*. An example of a Petri net consisting of three components is given in Figure 3. Here the transition T is shared between the components $C1$ and $C2$ and the place P is shared between $C2$ and $C3$. Before the actual analysis of the component-based model, we create a single net by merging the shared places and transitions. This is demonstrated in the lower part of Figure 3.

3.2 Translation of Little-JIL Primitives to CTAPN

We shall now present a translation of Little-JIL workflow constructs extended with explicit timing information into CTAPN. For each Little-JIL activity A we construct a *timed workflow net* (see e.g. [10, 17, 24–27]), a special form of a timed-arc Petri net, where

- there is exactly one *input place* called $start_A$ that has no input arcs, and
- a number of *output places* including end_A and optionally also other places[2] for modelling failed executions and exceptions such that all output places have no outgoing arcs.

[2] An extension to the standard workflow nets that only contain one output place.

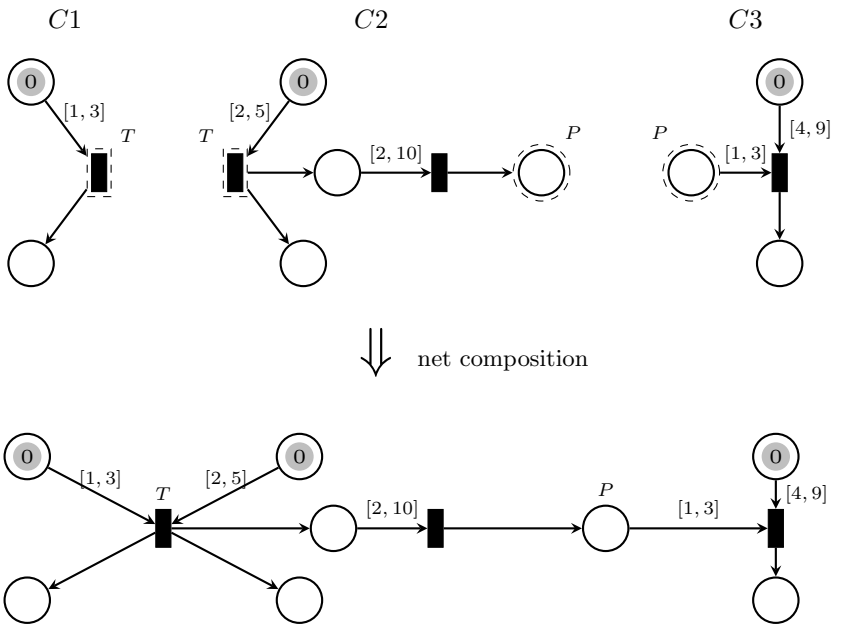


Fig. 3. Components $C1$, $C2$ and $C3$ and the Composed Net

For the external activity call (Figure 1(b)) the input and output places are shared and the activity is modelled in a separate component. In this way the whole net is decomposed into several components that are manageable (usually fit on one screen) and they are composed automatically before the verification of workflow properties is initiated.

We say that a net is *statically sound* if all places and transitions are on a path from the input place to some of the output places. A net is *dynamically sound* if during any execution starting with a marking having a token in the input place, we eventually reach a marking where one of the output places is marked. Note that a statically sound net is not necessarily dynamically sound. We show how to automatically verify dynamical soundness in Section 4. Statical soundness is guaranteed by the compositional construction of the workflow nets.

We say that $[L, U]$, where $L \leq U$, is the *execution interval* of a timed workflow net if L is the shortest and U the longest time needed to move a single token from the input place to some of the output places. Note that if a net is not dynamically sound, the execution interval is not well defined.

We shall now provide the details of the translation for building statically sound workflow nets for the Little-JIL constructs.

Atomic Activity. A timed workflow net corresponding to the atomic activity (Figure 1(a)) is depicted in Figure 4. The presence of the invariant $\leq U$ guarantees that the activity is executed no later than U time units since its initialization and the interval on the arc ensures that this does not happen earlier than at time L. Hence we get the following property.

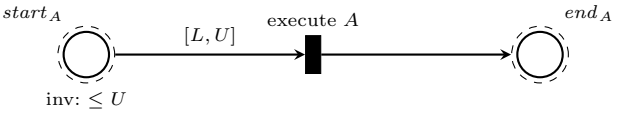


Fig. 4. Atomic activity

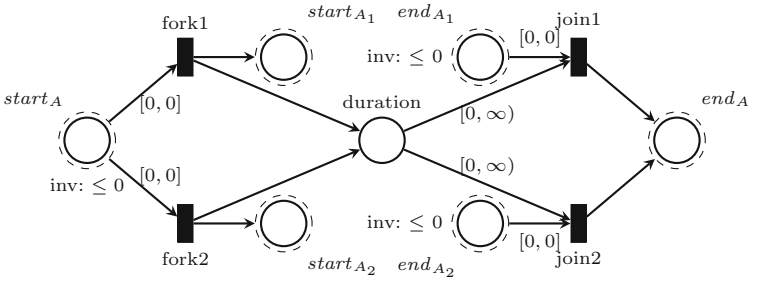


Fig. 5. Alternative activities

Property 1. *The execution interval of the net for atomic activity A is $[L, U]$.*

Note that the input and output places are shared so that they can be composed with other coordinating activities.

Alternative Activities. Figure 5 describes the choice between two alternatives of the Little-JIL diagram in Figure 1(d). The place duration is used to measure the current execution time (represented by the age of a token in this place) of the choice activity. A monitor (see Subsection 3.3) connected to the place duration can be used to detect a possible deadline violation.

Property 2. *Let $[L_1, U_1]$ and $[L_2, U_2]$ be the execution intervals of the activities A_1 and A_2, respectively. The execution interval of the net for alternative activities is* $[\min\{L_1, L_2\}, \max\{U_1, U_2\}]$.

Proof. From the initial marking that contains one token in place $start_A$ we have to, without any further delay due to the invariant ≤ 0, fire either the transition fork1 or fork2. This initiates the subnets for the activity A_1 or A_2. When these are finished, again due to the invariants ≤ 0 in places end_{A_1} and end_{A_2} we have to without any delay fire the transition join1 or join2. The lower and upper bounds of the execution interval are clearly the minimum and the maximum of the corresponding bounds for the activities A_1 and A_2. □

Parallel Activities. Parallel Little-JIL activities (Figure 1(e)) are modelled by the net in Figure 6. Here the subnets for the activities A_1 and A_2 are initiated concurrently by firing the transition fork. A duration place is added as before for the monitoring of the total duration of the parallel activities. There is added a mechanism that ensures that the place end_A is marked as soon as both parallel subtasks terminate.

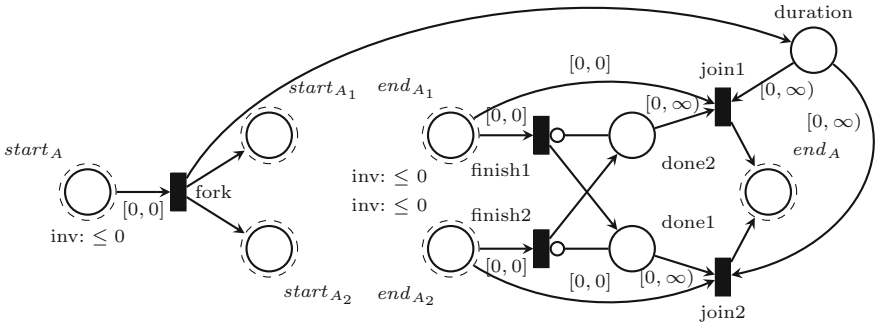


Fig. 6. Parallel activities

Property 3. *Let* $[L_1, U_1]$ *and* $[L_2, U_2]$ *be the execution intervals of the activities* A_1 *and* A_2*, respectively. The execution interval of the net for parallel activities is* $[\max\{L_1, L_2\}, \max\{U_1, U_2\}]$.

Proof. Notice that due to the age invariant ≤ 0 at place $start_A$ the two parallel activities are initiated without any delay. The construction connected to the end states for the two parallel activities ensures that a join happens as soon as both parallel activities mark their output places. Assume w.l.o.g. that the first subnet terminates first and marks the place end_{A_1}. Without further delay (imposed by the invariant ≤ 0 at place end_{A_1}) the transition finish1 must be fired; note that the transition join1 is not enabled. No other transitions are enabled until the subnet for A_2 terminates by placing a token into the place end_{A_2}. In this case the transition finish2 is not enabled due the the inhibitor arc and the fact that done1 already contains a token. Hence, without any further delay, the only option is to fire the transition join2 and mark the output place of the activity net for A (and at the same time consume the token from the place duration). □

Sequential Activities. Figure 7 shows how a diagram A consisting of two sequential Little-JIL activities A_1 and A_2 (described in Figure 1(f)) can be modelled as a timed workflow net. The point is that the first activity is activated immediately and the age of the token in the place duration1 can be used to measure the duration of the first activity. When the first activity is finished, the second activity is initiated without any delay. At the same time the token from duration1 is moved using the transport arcs to the place duration1+2 and its age in this place corresponds to the total duration of the first activity plus the current duration of the second activity. As mentioned before, a monitor can be attached to this place in order to check for the violation of deadlines.

Property 4. *Let* $[L_1, U_1]$ *and* $[L_2, U_2]$ *be the execution intervals of the activities* A_1 *and* A_2*, respectively. The execution interval of the net for sequential activities is* $[L_1 + L_2, U_1 + U_2]$.

Proof. Due to the presence of the invariant ≤ 0 in the place $start_A$, the first activity is initiated without any delay. When it is finished, thanks to the invariant

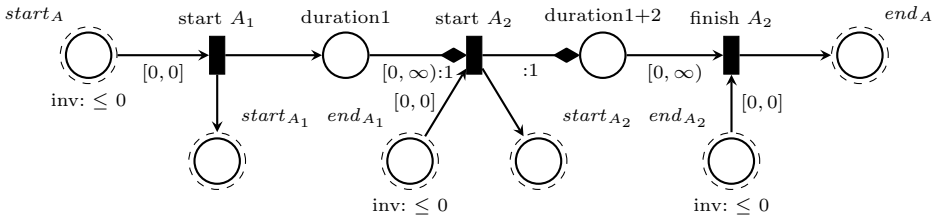


Fig. 7. Sequential activities

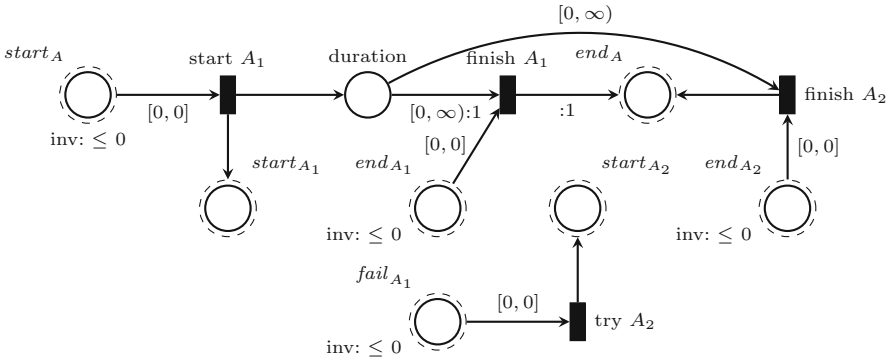


Fig. 8. Try A_1 else execute A_2

≤ 0 in place end_{A_1}, the second activity is initiated without any delay and because of the last invariant ≤ 0 at the place end_{A_2}, the whole workflow finishes in the time corresponding to the sum of the durations of the two activities. □

Try Activity. In the translation of the try construct (Figure 1(g)) presented in Figure 8, we first start with the execution of the first activity and if it ends successfully the whole try activity ends. If the first activity fails, we execute without any delay the second activity as an alternative. We assume that the net for the first activity A_1 contains a special output place $fail_{A_1}$ that gets marked whenever its execution fails.

Property 5. *Let* $[L_1, U_1]$ *and* $[L_2, U_2]$ *be the execution intervals of the activities* A_1 *and* A_2*, respectively. The execution interval of the net for try activity is* $[L_1, U_1 + U_2]$.

Proof. Clearly the first activity is called without any delay due to the invariant ≤ 0 at the place $start_A$ and within the interval $[L_1, U_1]$ the first subnet marks either end_{A_1} or $fail_{A_1}$. In the first case, again without any delay, the place end_A will be marked; in the second case the second activity is initiated without any delay and the execution of try stops as soon as this activity is finished. Then clearly the shortest execution time is L_1, assuming that the try activity succeeds on the first subactivity, and the longest one is $U_1 + U_2$. □

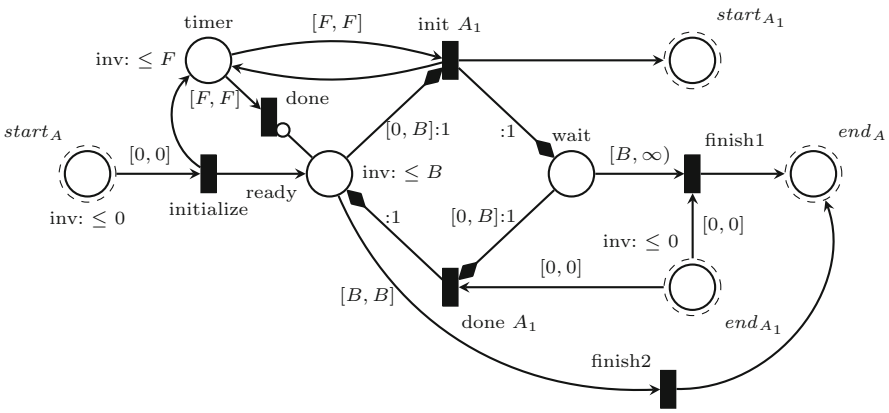


Fig. 9. Time-bounded iteration activity with frequency F and time-bound B

Time-Bounded Iteration. Figure 9 shows a net modelling the time-bounded iteration Little-JIL primitive from Figure 1(h). The iteration is parameterized by a frequency $F > 0$ and a time-bound $B \geq F$. This construct is typically used in healthcare workflows for repeated monitoring of a patient in a precisely given interval. The net enforces that the activity A_1 is initiated every F units of time (due to the invariant in the place timer) and that the last activation of A_1 happens no later then B time units from the activation of the iteration net.

After initializing the net and placing a token into the place ready and into the timer, we delay F time units and start the subactivity A_1. The token from the place ready is moved to the place wait while its age is being preserved; at the same time the age of the token in the timer is reset to zero. Once the activity A_1 is finished, we have to fire (with no delay) the transition done A_1 and move the token from wait to ready (unless the iteration is ended by firing finish1). Thanks to the transport arcs, the age of the token in ready measures the total execution time of the iteration activity.

The reader may observe that if the duration of the activity A_1 is longer than the frequency F then the token in the place timer will be removed by firing the transition done and once the place ready gets marked by firing the transition done A_1, the activation of A_1 at the frequency F is broken. We can detect such a situation via monitors.

Property 6. *Let $[L_1, U_1]$ be the execution interval of the activity A_1. The execution interval of the net for iteration activity is $[B, \max\{B, kF + U_1\}]$ where k is the largest integer such that $kF \leq B$.*

Proof. The lower bound of B time units is easy to prove as the age of the token that it moved by the transport arcs between the places ready and wait corresponds to the total duration of the iteration activity. The workflow can be terminated by firing either the transition finish1 or finish2 and both of them require a token of age at least B.

For the upper bound, we consider two cases. If $U_1 > F$ then during the slowest execution of A_1, the token in the place timer will be consumed and once the place ready gets marked, we can only wait until the total time reaches B and then fire the transition finish2. Hence the upper bound in this case is B.

If $U_1 \leq F$ then the transition init A_1 will be fired regularly after each F time units until the age of the token in the places ready or wait reaches the age B. This means that the last time the transition init A_1 can be fired is at the moment kF where k is the largest integer such that $kF \leq B$. After that we wait for the termination of the execution of A_1. In the worst scenario this takes U_1 time units, so the total execution time is $kF + U_1$ and if this exceeds the bound B then we are forced to fire immediately the transition finish1 and the longest execution time is $kF + U_1$. □

3.3 Additional Workflow Modelling Features

Exception Handling. We handle the exceptions (Figure 1(i)) in a similar way as the try construct presented in Figure 8. An atomic subactivity of A_1 (here not necessarily only a part of the sequential composition) can raise an exception E:name by placing a token into a new output place $exception_{\text{name}}$. The exception should be now caught by the first exception handler that covers its scope. As the nesting of exceptions in Little-JIL is always finite, we can create more copies of the place $exception_{\text{name}}$, one for each scope of the exception handler. The scopes are then updated dynamically during the computation of the net. As the scope information is finite, it is not surprising that we can remember its scope in this way. Nevertheless it is technically challenging to manually model multiple nested exceptions. In the case study we therefore used only a single nesting of exceptions that is easily manageable for a human modeller.

Shared Resources with Timing. Little-JIL provides also a mechanism for acquiring and releasing resources. Resources with exclusive access can be modelled in Petri nets in the standard way; we add here the additional option to measure the recovery time that has to pass from the time a resource was released until it can be acquired by another process. For example in our case study two nurses need to acquire a room for a transfusion and we have to guarantee exclusive access to the room. Moreover, after the room is released, some other activity (e.g. sterilization) must be performed before the room is ready for another patient. We can model this situation as depicted in Figure 10 via shared transitions acquire1 and release1 used by the first nurse and acquire2 and release2 used by the second one. It takes at least *minReady* and at most *maxReady* time units to prepare the room for another patient.

Monitors. We use different types of monitors (small nets attached to the workflow) in order to observe executions of events and their temporal dependencies. Figure 11(a) shows how the execution of the event checkID and beginInfusionOfBloodProduce can be registered by adding a component that will via shared transitions add a token to the place IDchecked resp. infusionStarted each time

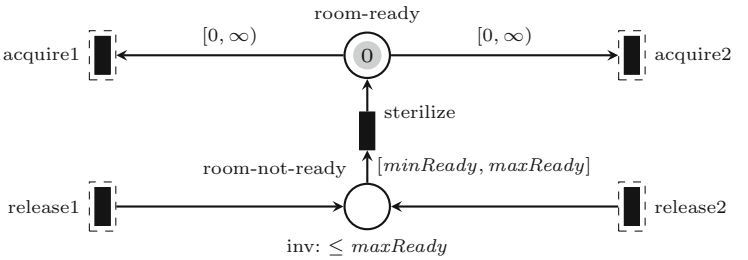
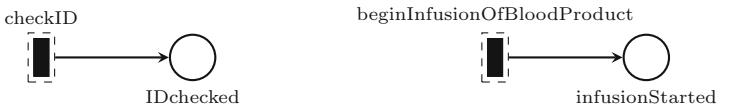
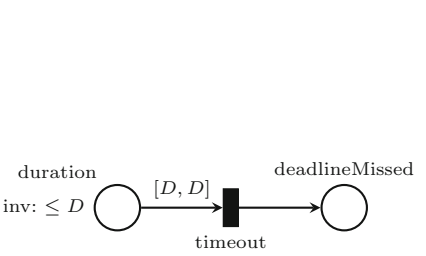


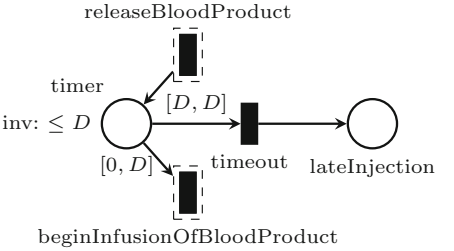
Fig. 10. Sharing of a resource (transfusion room example)



(a) Event registration



(b) Single-activity execution deadline



(c) Time-bounded response

Fig. 11. Monitors

the corresponding event is executed. Regarding temporal dependencies, observe that the nets for different Little-JIL primitives have a place (or several places) called duration. The duration place is marked by a token during the initialization of the activity and hence the age of the token represents the current execution time of the activity. We can add a monitor to such a duration place in order to check for the violation of the execution deadline D. The monitor is depicted in Figure 11(b) and it is clear that once the execution deadline D is reached, we are forced to mark the place deadlineMissed. Another type of monitor is given in Figure 11(c) where we can measure the response time between the events releaseBloodProduct and beginInfusionOfBloodProduct. Whenever the event releaseBloodProduct is not within D time units followed by the beginning of the transfusion, the place lateInjection gets marked.

4 Verification of the Blood Transfusion Case Study

Following the general algorithm presented in Section 3, we translated the blood transfusion case study described via Little-JIL diagrams into a component timed-arc Petri net and set up monitors for checking the crucial timing aspects of the

workflow. The full set of Little-JIL diagrams and the manually created CTAPN models are available at the URL `http://www.tapaal.net` in the download section. For example, for the single-patient model, the composed net consists of 140 places, 98 transitions and 243 arcs.

We used the model checker TAPAAL [9] for the actual editing and verification of the CTAPN model. The tool offers a user-friendly GUI with the support for a fully automatic verification of a subset of TCTL that includes the temporal operators EF φ (a marking satisfying the proposition φ is reachable), AG φ (the proposition φ is satisfied in any reachable marking), EG φ (there is a maximal computation invariantly satisfying φ) and AF φ (on every maximal computation φ eventually holds). The formula φ consists of a boolean combination of atomic propositions of the form $P \leq n$ and $P \geq n$ where P is a place and n is nonnegative integer, expressing the requirement that the number of tokens present in the place P is at most, resp. at least n.

We shall now present a selection of the verified queries for the blood transfusion case study; a full list of the queries is available within the TAPAAL model. Here `workflow_END` is the output place of the main workflow net and the other places are given in Figure 11. The place called `duration` corresponds to the overall duration of the whole workflow.

1. `AG (infusionStarted=0 or IDchecked>=1)` — ID is always checked at least once before the transfusion starts.
2. `EF (workflow_END=1 and deadlineMissed=0)` — The workflow can terminate within the deadline D (shortest execution time).
3. `AG (workflow_END=1 or deadlineMissed=0)` — In any scenario the workflow terminates within the deadline D (longest execution time).
4. `AG lateInjection=0` — Late injection never happens (the time from picking up the blood product until the transfusion starts is not more that D time units; if $D = 30$ then this is exactly the B.9 property from the benchmark).
5. `AF workflow_END=1` — Any maximal execution eventually reaches a marking where the whole workflow terminates (dynamical soundness).

Dynamical soundness implies that there are e.g. no deadlocks, no missing exception handlers and no improper use of the iteration activity. The temporal operator AF is a liveness operator and its verification is in general more demanding than the reachability and safety properties. We were able to positively verify this property for the whole workflow net in less than 10 second on a standard laptop; the other queries were positively verified in less than 1 second.[3] By varying the deadline D (declared as a constant in the net) we found out that the shortest execution time is 6 minutes (if patient disagrees with the transfusion), the longest execution time is 153 minutes and the longest time from picking up the blood product until its injection is 22 minutes (and hence the 30 minute expiration time imposed by B.9 property is met). Among the other properties

[3] All experiments were carried out by the native TAPAAL engines without using translations to UPPAAL timed automata that are also available in the tool.

we verified, we can e.g. mention that the shortest time of a successful transfusion is 118 minutes and we also successfully verified the property B.11 of the benchmark (patient is monitored during the transfusion every 15 minutes).

For two patients that share the same transfusion room, we confirmed in less than 40 seconds its dynamical soundness and verified that the longest execution time is 286 minutes. However, we found out that the B.9 property is broken. The tool provided an error trace showing a concrete execution of the workflow where the time from picking up the blood product until its injection exceeded 30 minutes. By examining the trace, we could easily find the reason: the pre-infusion activities allow in parallel to book the transfusion room and pick up the blood from the blood bank; as we might have to wait for the release of the transfusion room and its sterilization, the blood product can expire. This hints at the fact that, for more patients, these two activities have to be ordered sequentially.

A detailed comparison of the explicit state-space (discrete) verification methods and the DBM-based ones is beyond the scope of this paper but on the blood transfusion case study we can report that for the reachability properties the DBM-based methods were faster due to the higher constant sizes, whereas for liveness properties (dynamic soundness) the explicit methods were in this case considerably faster. A detailed comparison of the different verification methods is available in [3].

5 Conclusion

We have presented a general translation of medical healthcare workflows described in Little-JIL into component timed-arc Petri nets. As for any other workflow language, Little-JIL semantics can be conveniently given as a workflow Petri net via different constructions already described in the literature or via a direct translation into finite-state machines. The main contribution of our work is that we systematically model the *real-time aspects* of Little-JIL workflows and this allows us to use the tool TAPAAL for automatic verification of not only functional but also non-functional requirements as demonstrated on the blood transfusion case study.

The translation of the blood transfusion case study was performed manually but we are currently working on an automated tool for importing Little-JIL diagrams directly into TAPAAL. Another future research will focus on extending the property specification language and exploring how the technique will handle even larger case studies, including the possibility of direct code generation for automatic workflow coordination. On a different note, integrating human-specific aspects and ethical issues including security policies (like e.g. in OrBac [2]) into our approach is another challenge for the future work.

Acknowledgements. The work is partly supported by the projects GAVES, SAFEHR and EVGUI funded by the Macau Science and Technology Development Fund, and by MT-LAB, VKR Centre of Excellence. We thank to the anonymous reviewers for their detailed comments and suggestions.

References

1. Blood transfusion medical benchmark, https://collab.cs.umass.edu/groups/laser_library/wiki/daf17/Blood_Transfusion_Medical_Benchmark.html
2. Abou El Kalam, A., Baida, R.E., Balbiani, P., Benferhat, S., Cuppens, F., Deswarte, Y., Miège, A., Saurel, C., Trouessin, G.: Organization based access control. In: Policy 2003 (June 2003)
3. Andersen, M., Gatten Larsen, H., Srba, J., Grund Sørensen, M., Haahr Taankvist, J.: Verification of liveness properties on closed timed-arc petri nets. In: Kučera, A., Henzinger, T.A., Nešetřil, J., Vojnar, T., Antoš, D. (eds.) MEMICS 2012. LNCS, vol. 7721, pp. 69–81. Springer, Heidelberg (2013)
4. Bertolini, C., Schäf, M., Stolz, V.: Towards a Formal Integrated Model of Collaborative Healthcare Workflows. In: Liu, Z., Wassyng, A. (eds.) FHIES 2011. LNCS, vol. 7151, pp. 57–74. Springer, Heidelberg (2012)
5. Bolognesi, T., Lucidi, F., Trigila, S.: From timed Petri nets to timed LOTOS. In: 10th International Symposium on Protocol Specification, Testing and Verification, pp. 1–14. North-Holland, Amsterdam (1990)
6. Chen, B., Avrunin, G.S., Henneman, E.A., Clarke, L.A., Osterweil, L.J., Henneman, P.L.: Analyzing Medical Processes. In: ICSE 2008, pp. 623–632. ACM, New York (2008)
7. Chen, Z., Liu, Z., Ravn, A.P., Stolz, V., Zhan, N.: Refinement and verification in component-based model-driven design. Sci. Comp. Program. 74(4), 168–196 (2009)
8. Christov, S., Avrunin, G., Clarke, A., Osterweil, L., Henneman, E.: A benchmark for evaluating software engineering techniques for improving medical processes. In: Proceedings of the 2010 ICSE Workshop on Software Engineering in Health Care, SEHC 2010, pp. 50–56. ACM, New York (2010)
9. David, A., Jacobsen, L., Jacobsen, M., Jørgensen, K.Y., Møller, M.H., Srba, J.: TAPAAL 2.0: Integrated development environment for timed-arc Petri nets. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 492–497. Springer, Heidelberg (2012)
10. del Foyo, P.M.G., Silva, J.R.: Using time Petri nets for modelling and verification of timed constrained workflow systems. In: ABCM Symposium Series in Mechatronics, vol. 3, pp. 471–478. ABCM (2008)
11. Grando, M.A., Glasspool, D.W., Fox, J.: Petri Nets as a Formalism for Comparing Expressiveness of Workflow-Based Clinical Guideline Languages. In: Ardagna, D., Mecella, M., Yang, J. (eds.) BPM 2008 Workshops. LNBIP, vol. 17, pp. 348–360. Springer, Heidelberg (2009)
12. Hanisch, H.-M.: Analysis of place/transition nets with timed-arcs and its application to batch process control. In: Ajmone Marsan, M. (ed.) ICATPN 1993. LNCS, vol. 691, pp. 282–299. Springer, Heidelberg (1993)
13. Jacobsen, L., Jacobsen, M., Møller, M.H., Srba, J.: Verification of timed-arc Petri nets. In: Černá, I., Gyimóthy, T., Hromkovič, J., Jefferey, K., Králović, R., Vukolić, M., Wolf, S. (eds.) SOFSEM 2011. LNCS, vol. 6543, pp. 46–72. Springer, Heidelberg (2011)
14. Jensen, K.: Coloured Petri Nets: Basic concepts, analysis methods and practical use. Springer, Berlin (1996)
15. Ke, W., Li, X., Liu, Z., Stolz, V.: rCOS: a formal model-driven engineering method for component-based software. Frontiers of Computer Science in China 6(1), 17–39 (2012)

16. Lamport, L.: Real-time model checking is really simple. In: Borrione, D., Paul, W. (eds.) CHARME 2005. LNCS, vol. 3725, pp. 162–175. Springer, Heidelberg (2005)
17. Ling, S., Schmidt, H.: Time Petri nets for workflow modelling and analysis. In: IEEE International Conference on Systems, Man and, Cybernetics, vol. 4, pp. 3039–3044. IEEE (2000)
18. MacCaull, W., Rabbi, F.: NOVA Workflow: A Workflow Management Tool Targeting Health Services Delivery. In: Liu, Z., Wassyng, A. (eds.) FHIES 2011. LNCS, vol. 7151, pp. 75–92. Springer, Heidelberg (2012)
19. Mashiyat, A.S., Rabbi, F., Wang, H., MacCaull, W.: An automated translator for model checking time constrained workflow systems. In: Kowalewski, S., Roveri, M. (eds.) FMICS 2010. LNCS, vol. 6371, pp. 99–114. Springer, Heidelberg (2010)
20. Merlin, P., Faber, D.: Recoverability of communication protocols: Implications of a theoretical study. IEEE Trans. on Communications 24(9), 1036–1043 (1976)
21. Miller, K., MacCaull, W.: Model checking timed properties of healthcare processes. Journal of Software Maintenance and Evolution: Research and Practice 23(4), 245–260 (2011)
22. OMG. UML extensions for workflow process definition - request for proposal. OMG-document bom/2000-12-11 (December 2000)
23. Rabbi, F., Mashiyat, A.S., MacCaull, W.: Model checking workflow monitors and its application to a pain management process. In: Liu, Z., Wassyng, A. (eds.) FHIES 2011. LNCS, vol. 7151, pp. 111–128. Springer, Heidelberg (2012)
24. Salimifard, K., Wright, M.: Petri net-based modelling of workflow systems: An overview. European Journal of Operational Research 134(3), 664–676 (2001)
25. van der Aalst, W.: The application of Petri nets to workflow management. The Journal of Circuits, Systems and Computers 8(1), 21–66 (1998)
26. van der Aalst, W., van Hee, K.: Workflow Management: Models, Methods, and Systems. MIT Press (2002)
27. van der Aalst, W., Weske, M., Wirtz, G.: Advanced topics in workflow management: Issues, requirements, and solutions. Journal of Integrated and Process Science 7(3), 49–77 (2003)
28. WFMC. Workflow management coalition terminology and glossary (WFMC-TC-1011). Technical report, Workflow Management Coalition, Brussels (1996)
29. Wise, A.: Little-JIL 1.5 language report (UM-CS-2006-51). Technical report, University of Massachusetts, Amherst, MA (2006)