On Specifying and Visualising Long-Running Empirical Studies

Peter Y.H. Wong and Jeremy Gibbons

Computing Laboratory, University of Oxford, United Kingdom
{peter.wong,jeremy.gibbons}@comlab.ox.ac.uk

Abstract. We describe a graphical approach to formally specifying temporally ordered activity routines designed for calendar scheduling. We introduce a workflow model *OWorkflow*, for constructing specifications of long running empirical studies such as clinical trials in which observations for gathering data are performed at strict specific times. These observations, either manually performed or automated, are often interleaved with scientific procedures, and their descriptions are recorded in a calendar for scheduling and monitoring to ensure each observation is carried out correctly at a specific time. We also describe a bidirectional transformation between *OWorkflow* and a subset of Business Process Modelling Notation (BPMN), by which graphical specification, simulation, automation and formalisation are made possible.

1 Introduction

A typical long-running empirical study consists of a series of scientific procedures interleaved with a set of observations performed over a period of time; these observations may be manually performed or automated, and are usually recorded in a calendar schedule. An example of a long-running empirical study is a clinical trial, where observations, specifically case report form submissions, are performed at specific points in the trial. In such examples, observations are interleaved with clinical interventions on patients; precise descriptions of these observations are then recorded in a *patient study calendar* similar to the one shown in Figure 1(a). Currently study planners such as trial designers supply information about observations either textually or by inputting textual information and selecting options on XML-based data entry forms [2], similar to the one shown in Figure 1(b). However, the ordering constraints on observations and scientific procedures are complex, and a precise specification of this information is time consuming and prone to error. We believe the method of specification may be simplified and improved by allowing specifications to be *built formally and graphically, and visualised* as workflow instances.

Workflow instances are descriptions of a composition of activities, each of which describes either a manual task or an application of a program. One of the prominent applications of workflow technology is business processes modelling, for which the Business Process Modelling Notation (BPMN) [8] has been used as a modelling language. Recent research [9] has also allowed business processes

A. Vallecillo, J. Gray, A. Pierantonio (Eds.): ICMT 2008, LNCS 5063, pp. 76–90, 2008.

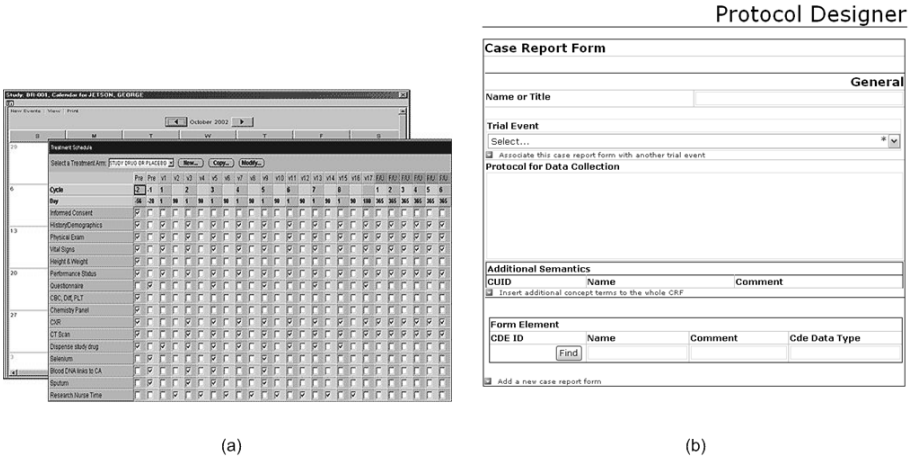

Fig. 1. (a) A screen shot of the patient study calendar [3], (b) XML-based data entry forms [2]

modelled as BPMN diagrams to be translated into executable processes in the Business Process Execution Language (WS-BPEL) [1], the "de facto" standard for web service compositions. Furthermore, BPMN has been given formal relative timed semantics [10]; these allow BPMN diagrams to be interpreted without ambiguity. BPMN, being a graphical language, lends itself to being used by domain specialists without computing expertise.

For example, we consider part of a cancer clinical trial, where there is a choice over two *case report form* submissions. The two reports are on tumour measurement or toxicity level. Both which to choice and when to report depend on the blood pressure of the patient concerned. Trial descriptions such as this could be specified as BPMN process and simulated as BPEL process for validation; in this paper, we present a customised workflow model *OWorkflow*, which is an extension to the CancerGrid trial model. Our notation allows empirical studies to be easily viewed and monitored through *study calendars*, while it is not intuitive to translate BPMN diagrams for calendar scheduling. We will revisit this example in Section 5.

This paper has two main contributions. Firstly, we introduce a generic observation workflow model *OWorkflow*, an extension of the workflow model implemented in the CancerGrid trial model [4], customised for modelling empirical studies declaratively. Secondly, we describe bidirectional transformation functions between *OWorkflow* and a subset of BPMN. While the transformation from BPMN to *OWorkflow* provides a medium for empirical studies to be specified graphically as workflows, transforming *OWorkflow* to BPMN allows graphical visualisation. Moreover, the BPMN descriptions of empirical studies may be translated into BPEL processes, whereby manual and automated observations may be simulated and executed respectively, and both of which can be monitored during the enactment of studies. Furthermore, BPMN has a formal semantics

and the transformation induces such behavioural semantics to *OWorkflow*. This means empirical study plans can now be formally specified, and interpreted without ambiguity.

The rest of this paper is structured as follows. We begin by giving a brief overview of BPMN in Section 2; a more detailed description of its abstract syntax may be found in our longer paper [11], and the complete definition of its relative timed semantics may be found in our other paper [10]. Section 3 describes the abstract syntax and the semantics of our workflow model *OWorkflow*. Here we only describe the semantics informally, even though a formal semantics has been defined via transformation to BPMN. Section 4 details the bidirectional transformation function between *OWorkflow* and the subset of BPMN by introducing BPMN constructs that are used as building blocks for modelling *OWorkflow*. We have implemented both the syntax of our observational workflow model and BPMN and the transformation functions in the functional programming language Haskell (see: `http://www.haskell.org`). Section 5 discusses how this transformation allows simulation and automation of empirical studies, and how formalisation has assisted the transformation process. Section 6 discusses related work and concludes this paper.

2 BPMN

In this section we give an overview of BPMN. For the purpose of specifying and simulating observational workflow *OWorkflow*, our implementation of BPMN states captures only a subset of BPMN, shown in Figure 2. This is a strict subset of the subset of BPMN formalised in our other paper [10]. We have implemented the corresponding syntax in Haskell. A fuller description of the syntax of this subset can be found in our longer paper [11].

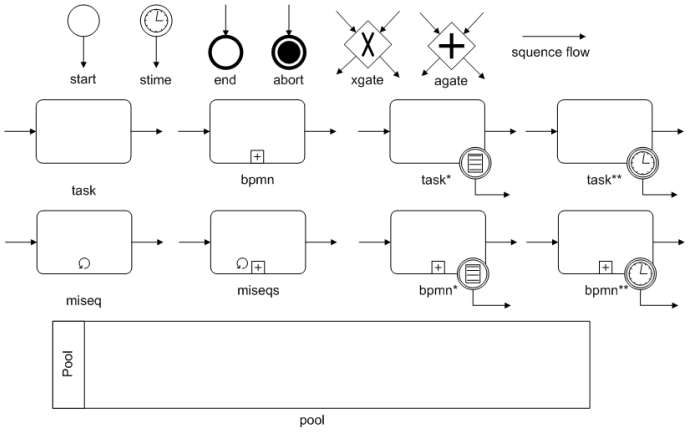

Fig. 2. States of BPMN diagram

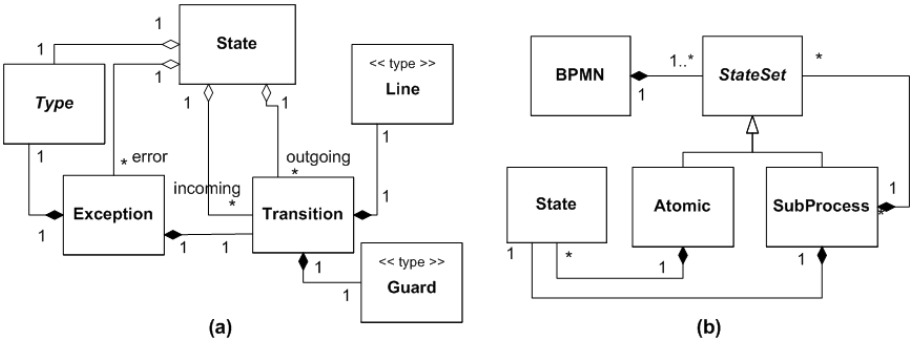

Fig. 3. Abstract syntax of (a) BPMN state and (b) BPMN diagram

States in our subset of BPMN [8] can either be events, tasks, subprocesses, multiple instances or control gateways, each linked by a normal sequence or an exception sequence flow. A normal sequence flow can be either incoming to or outgoing from a state and have associated guards; an exception sequence flow, depicted by the state labelled *task**, *bpmn**, *task*** and *bpmn***, represents an occurrence of error within the state. A sequence of flows represents a specific control flow instance of the business process. Figure 3(a) shows the abstract syntax of a BPMN state, where each state records its type, its lists of incoming and outgoing transitions, and its exception sequence flows as a list of pairs of exceptions types and corresponding transitions. Figure 3(b) shows the abstract syntax of a BPMN diagram, where each diagram is a collection of `StateSets`. Each `StateSet` defines either a list of non-subprocess states (`Atomic`), or a subprocess state (`SubProcess`), which records the type and sequence flows of the subprocess states, and a list of `StateSets` representing the subprocess's constituent states.

3 Abstract Syntax of Observational Workflow

In this section we describe the observation workflow model *OWorkflow*. This model generalises the clinical trial workflow model defined in the CancerGrid project [4]. Each workflow is a list of parameterised *generic* activity interdependence sequence rules, where each rule models the dependency between the prerequisite and the dependent observations. Figure 4(a) shows the abstract syntax of *OWorkflow*. Each sequence rule is implemented using the Haskell tuple type `EventSequencing`, which contains a single constructor `Event` and each observational workflow hence is a collection of sequence rules.

```
type OWorkflow = [EventSequencing]
data EventSequencing = Event ActId PreAct Condition Condition
                             (Maybe Obv) [RepeatExp] (Maybe Works)
```

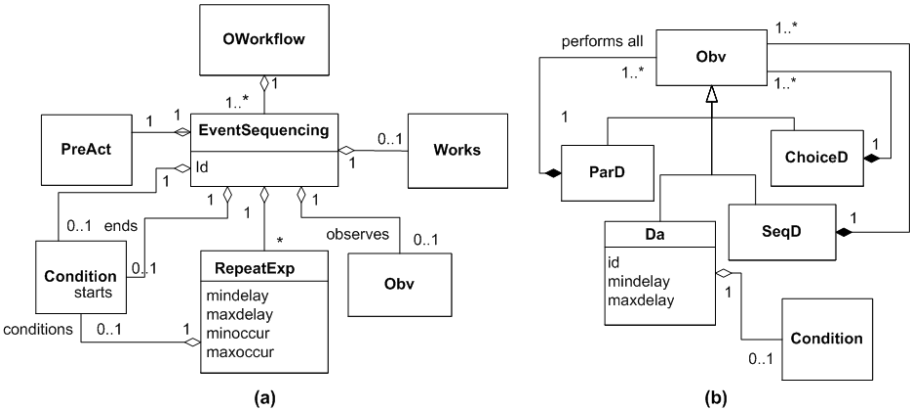

Fig. 4. Abstract syntax of (a) *OWorkflow* and (b) *observation group*

Each sequence rule is identified by a unique name of type `ActId` from the first argument of the constructor `Event`, and contains zero or more dependent observations. There are four reserved names of type `ActId` for identifying a start, a generic termination, a successful termination and an unsuccessful termination of a workflow execution. Each rule defines a structural composition of dependent observations of type `Maybe Obv`, in the fifth argument of the sequence rule. (A value of type `Maybe a` either contains a value of type `a`, or is empty.)

```
data Obv = ChoiceD [Obv] | ParD [Obv] | SeqD [Obv] | Da Act
type Act = (ActId,Duration,Duration,Condition,ActType)
```

We define a single dependent observation by the tuple type `Act`, whose first component is a unique name from a set of names `ActId` distinct from those which identify sequence rules. When performing dependent observations specified by each sequence rule, there exists a delay: a range with a minimum and a maximum duration, specified by the second and third component of `Act` of type `Duration`. Each duration records a string value in accordance with XML schema datatypes. For example in a clinical trial, the follow-up observation should be made between two and three months after all observations associated with the end of the treatment have been carried out. Each observation may either be a manual or an automated observation, denoted by the fifth component `ActType` of `Act`.

Each composition of observations defines an *observation group*, as shown in Figure 4. Figure 4(b) shows the abstract syntax of an *observation group*. Each observation group structurally conforms to Kiepuszewski's *structure workflow model* [5, Section 4.1.3]. The following inductive definition of compositional rules of an observation group follows from the definition of `Obv`:

1. If `obv :: Act` is a single observation, then `Da obv :: Obv` defines an observation group that yields to completion when the observation identified by `obv` has been made. We write `e :: T` to denote the expression `e` has type `T`.

2. Let `obv1,...,obvN :: Obv` be observation groups; their sequential composition `SeqD [obv1,...,obvN] :: Obv` also defines an observation group. Given an observation group `SeqD obvs`, observations are made sequentially starting at the head of `obvs`.
3. Similarly, let `obv1,...,obvN :: Obv` be n observations groups. An application of the choice operation over them `ChoiceD [obv1,...,obvN] :: Obv` defines an observation group, which structurally conforms the structure workflow model; it yields to completion when observations from one of the observation groups from the given list have been made. Likewise, `ParD [obv1,...,obvN]` yields to completion when observations from all of the observation groups have been made.
4. Nothing else defines an observation group.

Dependent observations are performed after the observations associated with the *prerequisite* sequence rules, identified by the data type `PreAct`, are completed. For example in a clinical trial the follow-up observation should be made after all observations associated with the end of the treatment have been carried out. A prerequisite is a collection of names that identifies preceding sequence rules, recorded in the second argument of `Event`. It is defined using the data type `PreAct`; we call each collection a *prerequisite rule group*.

```
data PreAct = All [PreAct] | OneOf [PreAct] | Pa ActId
```

The constructor `Pa` defines a single prerequisite rule by its argument, which yields to completion when all observations associated with the rule identified by the argument are made. The branching constructor `All` denotes synchronisation over its given list of prerequisite rule groups; this yields to completion when observations from all of the prerequisite rules groups from the given list have been made. The branching constructor `OneOf` denotes an exclusive merge over its given list of prerequisite rules groups; this yields to completion when observations from one of the prerequisite rules groups from the given list have been made.

Each sequence rule also defines a list, possibly empty, of *repeat* clauses described by the sixth argument, typed `[RepeatEx]`, of `Event`. Each clause specifies the condition, the minimum and the maximum numbers of iterations and the delay between iterations for the dependent observations of the sequence rule. These clauses are evaluated sequentially over the list after one default iteration of performing the rule's dependent observations.

```
type RepeatExp = (Duration,Duration,Int,Int,Condition)
```

Each clause, of type `RepeatExp`, contains a condition specified by the fifth component of type `Condition`. Our definition of `Condition` extends the *skip logic* used in the CancerGrid Workflow Model [4]. Specifically, its syntax captures expressions in conjunctive normal form.

```
data Condition = None | Nondeter | And [Alter]
data Alter = Alt [SCondition]
type SCondition = (Range,Property)
data Range = Bound RangeBound RangeBound | Emu [String]
data RangeBound = Abdate Duration | Abdec Float | Abint Int |
                  Rldate Property Duration | Rldec Property Float |
                  Rlint Property Int
```

Each condition `c :: Condition` yields a boolean value and is either empty (true), denoted by the nullary constructor `None`, nondeterministic denoted by the nullary constructor `Nondeter`, or defined as the conjunction of clauses, each of which is a disjunction of boolean conditions, of type `SCondition`. The type `SCondition` is satisfied if the value of specified property (typed `Property`) falls into the specified range (typed `Range`) at the time of evaluation. The specified property is a name that identifies a particular property in the domain of the empirical study and this corresponds the local property to the whole BPMN process [8, Section 8.6.1]. Note while our formal semantics of BPMN [10] allows behavioural process-based specifications and corresponding verifications for *OWorkflow*, it is at a level of abstraction in which we do not directly model the value of each properties.

The range may be an enumeration of values via the constructor `Emu`, or a closed interval of two numeric values via the constructor `Range` over two arguments of type `RangeBound`, which may be absolute or relative to a property.

Given a list of repeat clauses `res` defined in some sequence rule, evaluation begins at the head of the list. Each clause `res!!n`, where n ranges over `[1..(length res - 1)]`, it may be evaluated after the evaluation of the clause `res!!(n-1)` terminates. `res` terminates when `last res` terminates. (The operator `!!` denotes list indexing in Haskell.)

For example, the follow up sequence rule of a clinical trial might specify that follow up observations should be made every three months for three times after the default observations have been made, after which observations should be performed every six months for four times.

Each sequence rule might also include work units, recorded by the last argument of the constructor `Event`. Each work unit represents an empirical procedure such as administering a medical treatment on a patient in a clinical trial. In each sequence rule, the procedure defined by work units are interleaved with the rule's observations. Each collection of work units is defined by the data type `Works` and is called *work group*.

```
data Works = ChoiceW [Works] | ParW [Works] | SeqW [Works] | Wk Work
```

The type `Work` records a unique name that identifies a particular empirical procedure. Our definition of work group also structurally conforms to Kiepuszewski's structure workflow model, and both its abstract syntax and compositional rules are similar to those of observation groups.

Finally the third and fourth arguments of a sequence rule are two conditional statements, each of type `Condition`. While the third argument defines the condition for enacting the sequence rule, the fourth argument defines the condition for interrupting the enactment of the sequence rule.

4 Transformation

In this section we describe the bidirectional transformation between observation workflows of type `OWorkflow` and their corresponding subset of BPMN diagrams. Specifically we have implemented a total function transforming `OWorkflow` to `BPMN` and its inverse, a partial function transforming a subset of `BPMN` to `OWorkflow`.

```
w2b :: OWorkflow -> BPMN
b2w :: BPMN -> OWorkflow
```

For reasons of space we only informally describe the transformation of a single sequence rule to its corresponding BPMN subprocess state by explaining the transformation over each of the components that make up the 7-tuple of a sequence rule. We describe the transformation of individual components by introducing some building blocks in BPMN, which may be mapped to those components. A fuller description of the transformation may be found in our longer paper [11]. We stress that these transformation are completely automated.

4.1 Observation

Figure 5 shows an *expanded* BPMN subprocess state depicting a single dependent observation, of type `Act`. An observation may be performed after a delay ranging from the minimum to the maximum duration, provided that its associated condition is satisfied. The delay range is graphically modelled by first modelling minimum duration as the *stime* state (timer start event), and then modelling the duration ranges from the elapse of the minimum duration to the maximum duration using a task state which halts for an unknown duration, with an expiration exception flow, of which the expiry duration is the difference between maximum and minimum durations of the delay. We use a *xgate* (exclusive choice) decision gateway state for accepting either the task state's outgoing transition or its expiration exception flow.

The decision gateway is then followed by a task state, which models the actual observation itself and is identified by applying the function `idToTName` to the identifier of the observation being mapped.

```
idToTName :: ActId -> TaskName
```

An *end* state follows immediately for terminating the execution of the subprocess. The subprocess itself has one incoming and one outgoing transition,

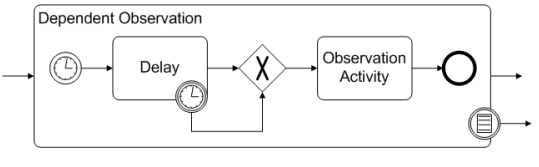

Fig. 5. A BPMN subprocess state depicting a single observation

denoted as the *outermost* incoming and outgoing transitions respectively. We have implemented the function `mkAct` to transform the subprocess state modelling an observation of type `Obv`, encapsulating a tuple `Act` describing a single observation via the constructor function `Da`. We have also implemented the function `mkDpt` to transform the tuple `Act` describing a single observation to a BPMN subprocess modelling that observation.

```
mkAct :: StateSet -> Obv
mkDpt :: Act -> Line -> ([StateSet],Line)
```

4.2 Groups

Each sequence rule contains zero or more observations and work units. Whereas the transformation of a single observation has been described in Section 4.1, each work unit is modelled as a task state, of which the name that identifies the task is obtained by applying the function `workToTask` on the unique identifier of the work unit. Conversely, the function `taskToWork` is defined to map a task state name to the unique name of the work unit it models. One or more observations compose into an observation group, which has been defined inductively in Section 3. Similarly one or more work units compose into a work group. Due to the conformity of both types of compositions to the structured workflow model [5] as mentioned in Section 3, we have generalised the notion of *group* and here we describe the transformation between a group and its corresponding BPMN subprocess state, which may be applied to both observation group and work group.

An example BPMN subprocess modelling a group is shown in Figure 6. It shows a BPMN subprocess state describing an observation group defined by the constructor `ParD` over a list of two observations, each defined by the constructor `Da`. A similar BPMN subprocess state may be defined to describe a work group. We describe informally the transformation rules for a group as follows:

1. Given group *go* defined by the constructor over a single activity *sa*, specifically `Da` applied over a single observation for an observation group and `Wk` applied over a single work unit for a work group respectively, we transform *sa* according the type of the activity, for an observation, the transformation rule has been described in Section 4.1, and a work unit is simply represented

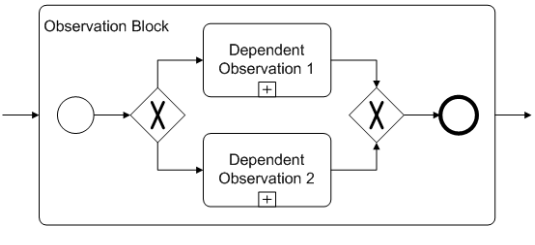

Fig. 6. A BPMN subprocess state depicting an observation group

by a task state, of which the task is identified by the name of the work unit. We use *sa*'s outermost incoming and outgoing transitions as *go*'s outermost incoming and outgoing transitions.

2. Given a group *go* defined by either a choice or a parallel constructor over a list of n groups, specifically `ChoiceD` and `ChoiceW` or `ParD` and `ParW` applied over a list of observation groups and work groups respectively, where $n \geq 1$, the corresponding BPMN states are either two *xgate* decision gateways for choice construction or two *agate* decision gateways for parallel construction. The first of these has one incoming transition, denoted as the *go*'s outermost incoming transition, and n outgoing transitions, each matching the outermost incoming transition from one of the n groups, and the other one has n incoming transitions, each matching the outermost outgoing transition from one of the n groups, and one outgoing transitions, denoted as the *go*'s outermost outgoing transition. The transformation of the n groups are defined recursively.
3. Given an observation group *go* defined by the sequential constructor over a list of n groups, specifically `SeqD` and `SeqW` over a list of observation groups and work groups respectively, where $n \geq 1$, the outermost outgoing transition of each group is matched by the outermost incoming transition of its next group. The outermost incoming transition of the first group defines the outermost incoming transition of *go*, and the outermost outgoing transition of the last group defines the outermost outgoing transition of *go*.

We have implemented the function `getObv` to transform the subprocess state describing an observation group to an observation group of type `Obv`.

```
getObv :: StateSet -> Obv
getInv :: StateSet -> Works
```

Similarly, we have implemented the function `getInv` to transform a work group of type `Works`. Conversely, we have implemented the functions `extObv` and `extWks` to transform an observation group of type `Obv` and a work group of type `Works` to a subprocess state describing that group, respectively.

```
extObv :: Line -> Obv -> ([StateSet],Line)
extWks :: Line -> Works -> ([State],Line)
```

4.3 Repeat Clauses

Figure 7 shows a BPMN subprocess modelling a single repeat clause. According to the semantics of a repeat clause, each repeat clause in a sequence rule repeats all dependent observations defined in that rule; the number of repetitions from each clause ranges between a minimum and a maximum value, and there is a delay, ranging between a minimum and a maximum duration, before each repetition can start. We model the delay range of a repeat clause graphically according to the transformation rules defined for a single observation in Section 4.1.

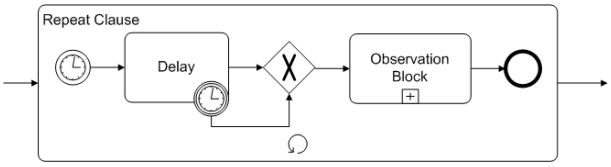

Fig. 7. A BPMN subprocess state depicting a repeat clause

We model each repeated observations as a subprocess state according the transformation of groups in Section 4.2. The subprocess, which defines the repeat clause, is a multiple instance *miseqs* state, and it has one incoming and one outgoing transition, denoted as the *outermost* incoming and outgoing transitions respectively. The multiple instance subprocess state is implemented by the Haskell type `Miseqs` which takes an integer value to specify the maximum number of repetitions and a condition to specify the conjunction of the minimum number of repetitions required and the clause's conditional statement.

A list of repeat clauses is therefore transformed iteratively over each clause starting from head of the list, similar to the transformation of a group for some sequential constructor described in the Rule 3 in Section 4.2. Figure 8 shows a BPMN subprocess state representing a list of two repeat clauses. Individual repeat clause is shown as collapsed subprocess state.

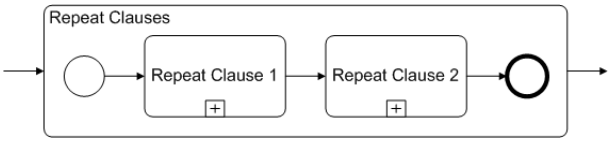

Fig. 8. A BPMN subprocess state depicting a list of two repeat clauses

4.4 Sequence Rules

Figure 9 shows a BPMN subprocess state representing a single sequence rule. The subprocess state is defined by three other subprocess states, collapsed in the figure, which model observations, work units and repeat clauses defined in the sequence rule. A sequence rule is enacted by first performing all its observations once, modelled by the subprocess *observation block*, after which the list of repeat clauses, modelled by the subprocess state *repeat clauses* is evaluated. As explained in Section 3, work units are empirical procedures and their executions are interleaved with their corresponding observations, hence we use an *agate* decision gateway state to initialise both observations and work units. We do not constrain how work units are interleaved with observations as our current workflow model focuses on the specification of observations, therefore it solely depends on the study planners. Note if no work unit is defined in the sequence rule,

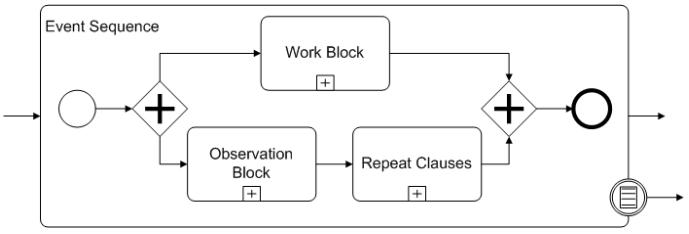

Fig. 9. A BPMN subprocess representing a single sequence rule

the corresponding subprocess will not have *agate* states and will be represented by a sequential composition of the *observation block* and *repeat clauses* states.

Finally we associate a *conditional* exception sequence flow with each subprocess state to model the enacting and the interrupting conditions of the sequence rule. A detailed description can be found in our longer paper [11].

5 On Simulation, Automation and Formalisation

In this section we discuss briefly the application of business process management technique to empirical studies. We describe informally, via a simple example, how modelling empirical studies in BPMN allows their study plans to be simulated and partially automated by translating the BPMN diagrams into executable BPEL processes. We also discuss how modelling empirical studies in BPMN has consequently induced a formal behavioural semantics upon our observation workflow model and hence removed ambiguities in both the transformations and interpretation of *OWorkflow*.

As useful as it is to visualise and formally specify a complete study plan, it is also beneficial to validate the plan before its execution phase, especially if the study has a long running duration, since it is undesirable to run into an error three months into the study! One method of validating a study is by simulation. When considering either simulating or automating a portion of a study, we assume the observations specified in that portion can be appropriately simulated or automated; an observation might define the action of recording a measurement from a display interfacing with a software application or submitting a web form to a web service for analysis. For example, the following specifies a simplified observation group, modelling a choice over two different case report form submissions in a clinical trial described briefly in Section 1.

```
ChoiceD [Da (Id "Tumour Measurement Report", Dur "P1D",Dur "P1D",
             Ands [Ors [(Emu ["low"],"blood pressure")]],Manual),
         Da (Id "Toxicity Review", Dur "P1D",Dur "P1D",
             Ands [Ors [(Emu ["high"],"blood pressure")]],Manual)]
```

While submitting a report form is a manual task, due to the transformation, it is possible to simulate this action by translating its corresponding BPMN subprocess state into the corresponding sequence of BPEL activities:

```
<switch>
 <case condition="getVariableData('blood pressure') == high">
  <wait for="PT1M"><operation name="sendToxicityReview">
   <input message="toxicityMessage" /></operation></wait></case>
 <case condition="getVariableData('blood pressure') == low">
  <wait for="PT1M"><operation name="sendTumourReport">
   <input message="tumourMessage" /></operation></wait></case>
</switch>
```

where each *wait* activity is an invocation upon the elapse of a specified duration. Since the derived BPEL process is for simulation, we scale down the specified duration of each observation. Note each invocation in a BPEL process is necessarily of a web service; if the specified observation defines an action to invoke a web service, e.g. uploading a web form, the translated BPEL operation will also be invoking that web service, and otherwise, for simulation purposes, a "dummy" web service could be used for merely receiving appropriate messages. Similarly, partial automation is also possible by translating appropriate observations into BPEL processes which may be executed during the execution phase of the study.

In recent work, BPMN has been given a formal relative timed semantics; in particular one has been defined in the process algebra CSP [10]. By defining a transformation function between `OWorkflow` and BPMN, it has automatically induced a behavioural semantics for `OWorkflow`. For example, Figure 10 shows

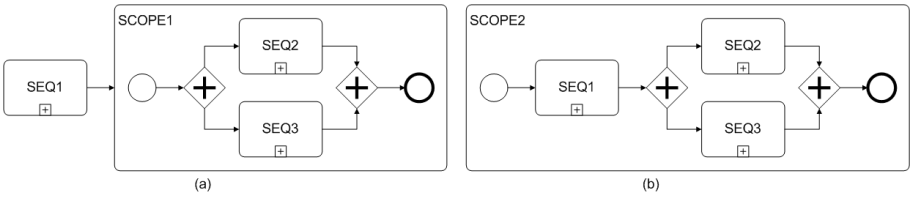

Fig. 10. Two BPMN diagrams modelling semantically equivalent observation workflow

two different BPMN diagrams partially, each modelling the same observation workflow described below, omitting description of observations and work units.

```
[Event (Id "SEQ1") (Pa START), Event (Id "SEQ2") (Pa (Id "SEQ1")),
 Event (Id "SEQ3") (Pa (Id "SEQ1")),
 Event NORMAL_STOP (All [Pa (Id "SEQ2"),Pa (Id "SEQ3")])]
```

Although applying the function `w2b` over this *OWorkflow* definition will yield the diagram in Figure 10(a), one would like to know if applying the function `b2w` over the two diagrams will yield the same *OWorkflow* definition. The formal semantics of BPMN in CSP [10] allows us to show that these two diagrams are in fact semantically equivalent, by model checking the following failures refinement assertions: $PLAN1 \sqsubseteq_F PLAN2 \wedge PLAN2 \sqsubseteq_F PLAN1$ where $PLAN1$

and $PLAN2$ are CSP processes describing the semantics of the partial BPMN diagrams in Figure 10. This means both $PLAN1$ and $PLAN2$ have the same behaviour and yields the same *OWorkflow* definition. Our semantic definition also allows formal verification of observation workflow against behavioural specifications, an example of which may be found in our longer paper [11].

6 Conclusion

Specifications of long running empirical studies are complex; the production of a complete specification can be time consuming and prone to error. We have described a graphical method to assist this type of specification. We have introduced an observation workflow model *OWorkflow* suitable for specifying empirical studies, which then can be populated onto a calendar for scheduling, and described bidirectional transformations, which allow empirical studies to be constructed graphically using BPMN, and to be simulated and partially automated as BPEL processes. The transformation also induces a behavioural semantics upon *OWorkflow*, and we have described the use of the semantics to remove ambiguity in the transformation process.

To the best of our knowledge, this paper describes the first attempt to apply graphical workflow technology to empirical studies and calendar scheduling, while large amounts of research have focused on the application of workflow notations and implementations to "in silico" scientific experiments. Notable is Ludäscher et al.'s Kepler System [6], in which such experiments are specified as a workflow graphically and fully automated by interpreting the workflow descriptions on a runtime engine. On the other hand we employ BPMN as a graphical notation to specify and graphically visualise experiments and studies that are typically long-running and in which automated tasks are often interleaved with manual ones. Studies such as clinical trial would also include "in vivo" intervention. Furthermore, our approach targets studies that are usually recorded in a calendar schedule to assist administrators and managers. Similarly, research effort has been directed towards effective planning of *specific types* of long running empirical studies, namely clinical trials and guidelines. Notable is Modgil and Hammond's Design-a-Trial (DaT) [7]. DaT is a decision support tool for critiquing the data supplied specifically for randomized controlled clinical trial specification based on expert knowledge, and subsequently outputting a protocol describing the trial. DaT includes a graphical trial planner, which allows description of complex procedural contents of the trial. To ease to complexity of protocol constructions, DaT uses macros, common plan (control flow) constructs, to assist trial designers to construct trial specification.

Future work will include extending our observation workflow model for more detail specifications of work units, such as temporal and procedural information, thereby allowing study plans to be verified against specifications of the relationship between work units and observations.

Acknowledgments

This work is supported by a grant from Microsoft Research. The authors are grateful to Radu Calinescu for many insightful discussions during this work. The authors would also like to thank anonymous referees for useful suggestions and comments.

References

1. Business Process Execution Language for Web Services, Version 1.1. (May 2003), http://www.ibm.com/developerworks/library/ws-bpel
2. Calinescu, R., Harris, S., Gibbons, J., Davies, J., Toujilov, I., Nagl, S.: Model-Driven Architecture for Cancer Research. In: Software Engineering and Formal Methods (September 2007)
3. Clinical Trials Management Tools. University of Pittsburgh, http://www.dbmi.pitt.edu/services/ctma.html
4. Harris, S., Calinescu, R.: CancerGrid clinical trials model 1.0. Technical Report MRC/1.4.1.1, CancerGrid (2006), http://www.cancergrid.org/public/documents
5. Kiepuszewski, B.: Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows. PhD thesis, Queensland University of Technology, Brisbane, Australia (2002)
6. Ludascher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger-Frank, E., Jones, M., Lee, E., Tao, J., Zhao, Y.: Scientific Workflow Management and the Kepler System. In: Concurrency and Computation: Practice & Experience, Special Issue on Scientific Workflows (to appear, 2005)
7. Modgil, S., Hammond, P.: Decision support tools for clinical trial design. Artificial Intelligence in Medicine 27 (2003)
8. Object Management Group. BPMN Specification (February 2006), http://www.bpmn.org
9. Ouyang, C., van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M.: Translating BPMN to BPEL. Technical Report BPM-06-02, BPM Center (2006)
10. Wong, P.Y.H., Gibbons, J.: A Relative-Timed Semantics for BPMN, Submitted for publication. Extended version (2008), http://web.comlab.ox.ac.uk/oucl/work/peter.wong/pub/bpmntime.pdf
11. Wong, P.Y.H., Gibbons, J.: On Specifying and Visualising Long-Running Empirical Studies (extended version) (2008), http://web.comlab.ox.ac.uk/oucl/work/peter.wong/pub/transext.pdf