ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Business Process Management

Formal Properties

Declarative, Open Languages

Federico Chesani
Department of
Computer Science and Engineering
University of Bologna

Credits and sources

These slides have been partly inspired by:

- Mathias Weske, *Business Process Management*, Second Edition, Springer, 2012 https://bpm-book.com/BpmBook/WebHome

- https://academic.signavio.com/p/login

- https://www.draw.io/

Formal Properties of Processes

Why formal properties? Which formal properties?

- If a process model guarantees a property, all process instances will ensure that property
- It is a way to ensure that **by design** our process instance will exhibit a certain behaviour

- Two different types of properties:
    - Structural, control-flow related properties
    - Domain related properties

- Will my process ever reach a certain outcome?
- Under which circumstance will my process reach a certain outcome?
- Is it the case that some path will be never walked down?
- Will my process ever end?
- Will always be the case that if *XXX*, then also *YYY* ?

Which formal properties?

Structure, control-flow related properties:

- Object Lifecycle
- Structural Soundness
- Soundness
- Relaxed Soundness
- Weak Soundness
- Lazy Soundness

Object Lifecycle Conformance

- Process instances act on information objects
- Information objects capture a set of information that are related to some aspect of my process instance
- Information objects should be manipulated (updated, modified) following a set of rules:
    - Rules about the data (e.g., integrity constraints)
    - Rules about how data evolve, i.e. about their **behaviour**

- Information objects have a **lifecycle**
- Such lifecycle define which **states**, **how**, and **when** an information objects can pass through
- It is a data-related perspective on procedural aspects

Object Lifecycle Conformance - Example

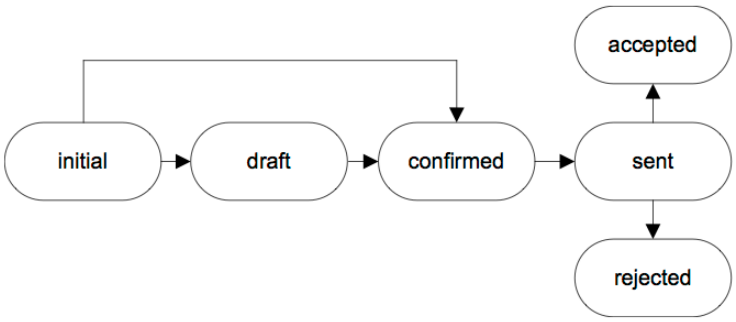
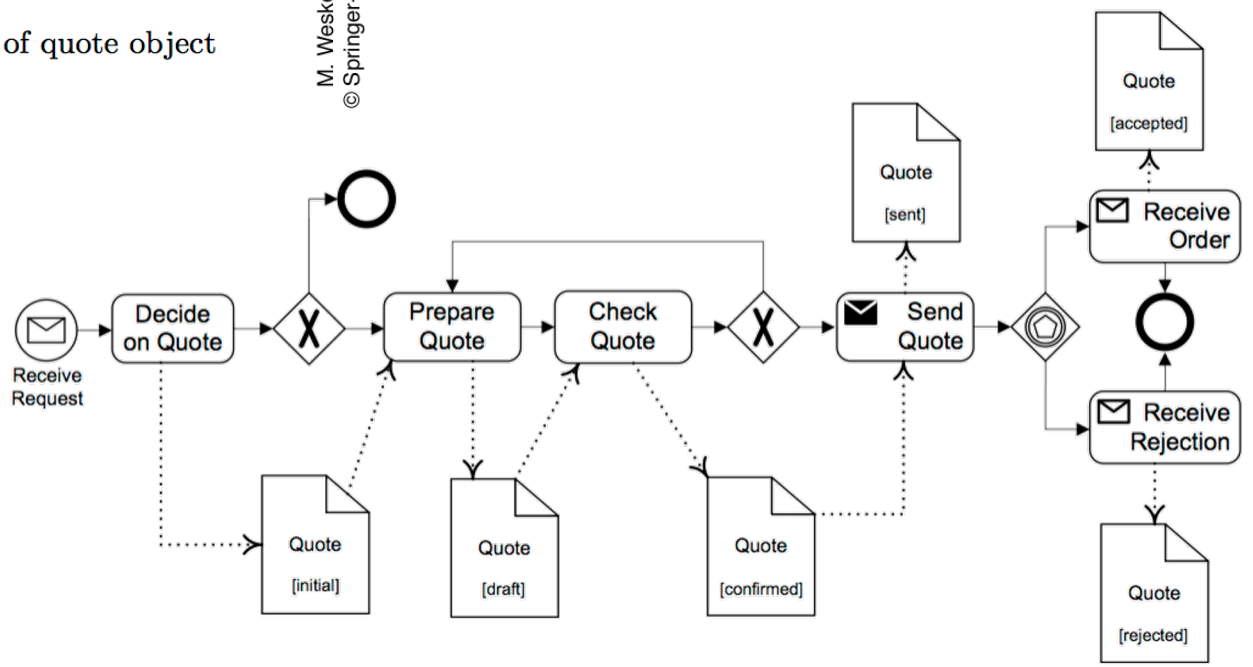


Fig. 6.3. Object lifecycle of quote object



Object Lifecycle Conformance - Example

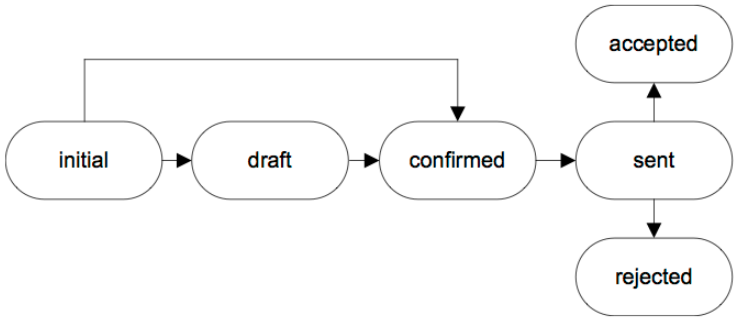
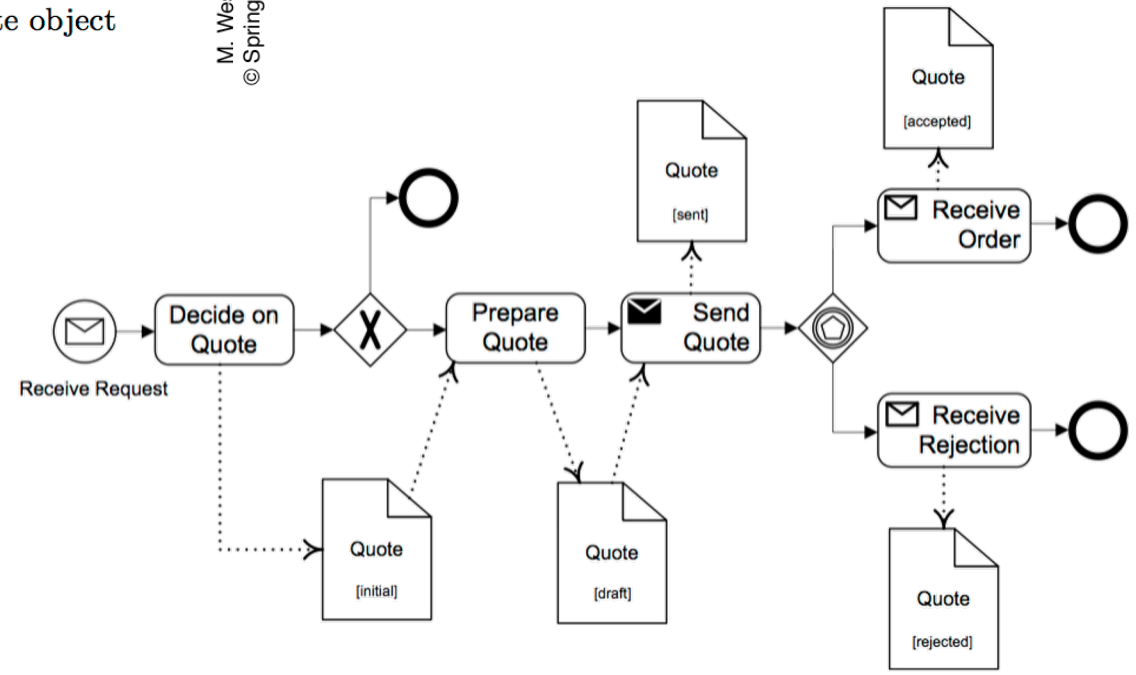


Fig. 6.3. Object lifecycle of quote object



Structural Soundness

- Petri Nets are quite a powerful formalism…
- … but they allow strange models (strange form the BP viewpoint)
- **Structural soundness** has been already introduced with workflow nets:
  - There is only one distinguished place *i* (*initial place)* that has no incoming edges
  - There is only one distinguished place *o* (*final place)* that has no outgoing edges
  - Every place and every transition is located on a path from the initial place to the final place

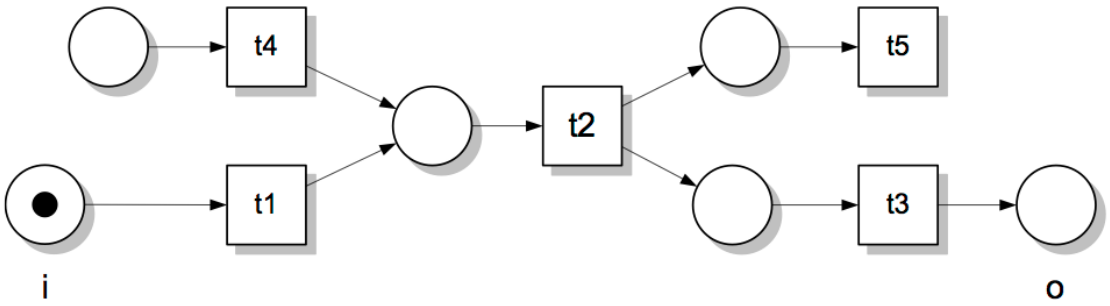


Fig. 6.6. Petri net with dangling places and dangling transitions

Soundness – Why?

- Structural Soundness might not be enough

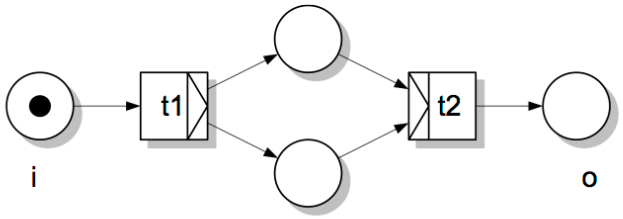


Fig. 6.7. Workflow net with deadlock

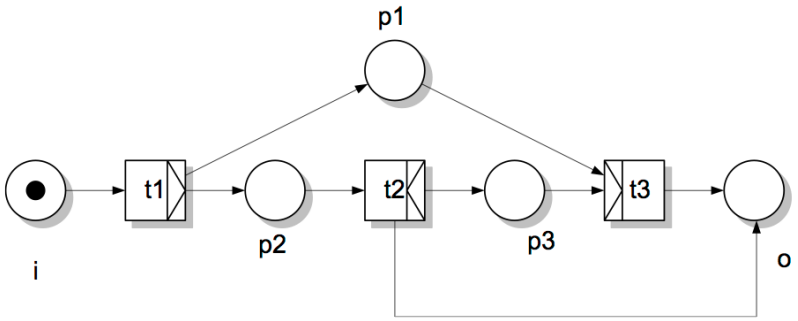


Fig. 6.9. Workflow net with deadlock/remaining tokens

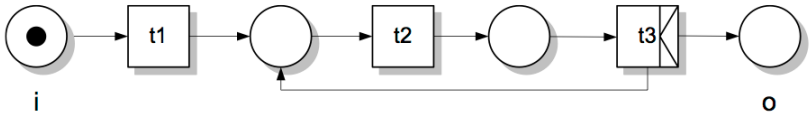


Fig. 6.8. Workflow net with livelock

Soundness – Few Definitions

Let PN = (P,T,F) be a workflow net, i ϵ P be its initial place, o ϵ P its final place, and M, M' markings.

- [i] is the state in which there is exactly one token in place I and no token in any other place
- [o] is the state in which there is exactly one token in place o and no token in any other place

- $M \geq M' \quad iff \quad \forall p \in P \quad M(p) \geq M'(p)$
- $M > M' \quad iff \quad M \geq M' \quad \wedge \quad \exists p \in P : M(p) > M'(p)$

Soundness – Definition

A workflow system (PN, i) with a workflow net PN=(P,T,F) is **sound** if and only if:

- For every state M **reachable** from state [i] there exists a firing sequence leading from M to [o], that is:

$$\forall M \left([i] \xrightarrow{*} M \right) \Longrightarrow \left(M \xrightarrow{*} [o] \right)$$

- State [o] is the only state reachable from state [i] with at least one token in place o, that is

$$\forall M \left([i] \xrightarrow{*} \mathrm{M} \wedge M \geq [o] \right) \Longrightarrow (M = [o])$$

- There are no dead transitions in the workflow net in state [i], that is

$$(\forall t \in T) \;\; \exists M, M' : \;\; [i] \xrightarrow{*} M \xrightarrow{t} M'$$

Soundness – Checking it through reachability

- Soundness can be checked through a reachability analysis
- How? First step, a reachability graph is constructed

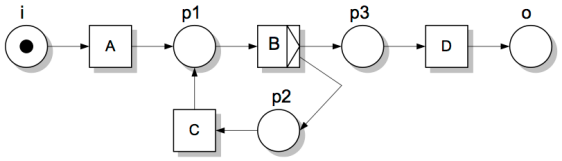


Fig. 6.11. Workflow net with loop

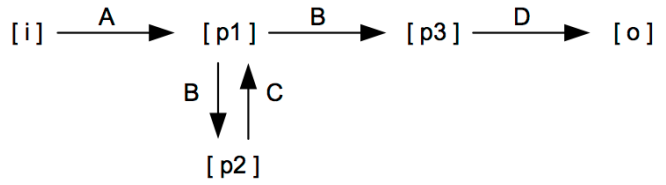


Fig. 6.12. Reachability graph of workflow net shown in Figure 6.11

Then, check that:

- For each state reachable from [i], there is a continuation to [o]
- Check that [o] is the only state reachable from [i], with one token in place o.
- All transitions participate in (at least) one execution that starts from[i] and terminates in [o]. I.e., looks for at least a path for each transition

Soundness – Checking it through reachability – Which problem?

- Soundness can be checked through a reachability analysis

- How? First step, a reachability graph is constructed

Then, check that:

- For each state reachable from [i], there is a continuation to [o]
- Check that [o] is the only state reachable from [i], with one token in place o.
- All transitions participate in (at least) one execution that starts from[i] and terminates in [o]. I.e., looks for at least a path for each transition

The reachability graph suffer of the state explosion issue.

This means that for real world applications is not practicable.

Soundness and Soundness Theorem

A different approach proceed as follows:

- Take a Workflow net PN
- Create a new workflow net PN', by adding a transition t^* , and link o to t^* and t^* to i

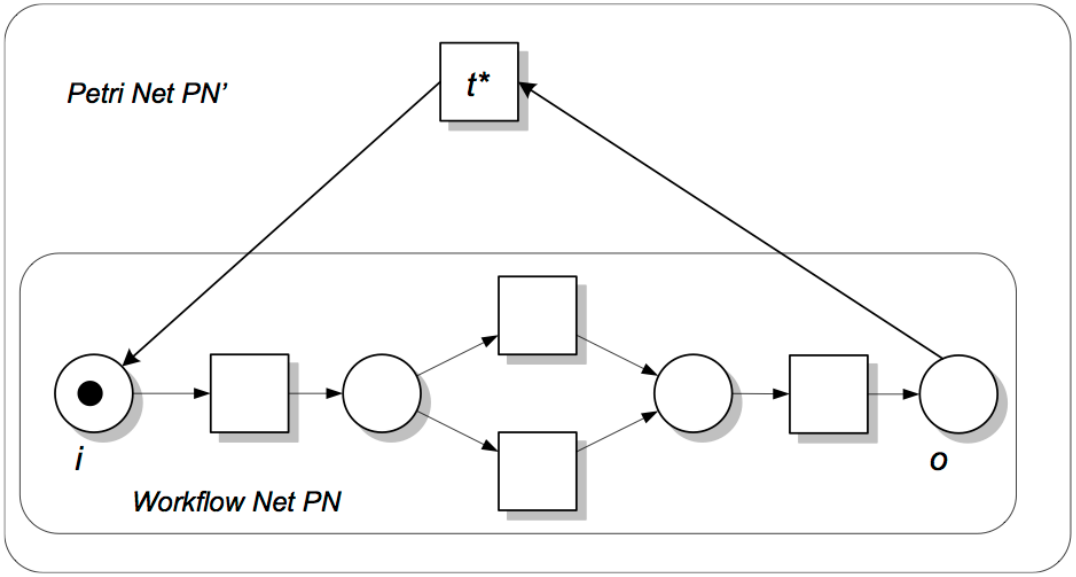


Fig. 6.13. Workflow net PN and Petri net PN', illustrating soundness theorem

Soundness and Soundness Theorem

Theorem

Let PN=(P,T,F) be a workflow net, and $t^* \notin T$. PN is **sound** if and only if (PN', i), such that PN'=(P', T', F'), $T' = T \cup \{t^*\}$, and $F' = F \cup \{(o, t^*), (t^*, i)\}$, is **live** and **bounded**.

A Petri Net is **live** (aka live$_1$) if for each transition t there is a firing sequence starting from [i] that actives t.

A place in a Petri net is called ***k-bounded*** if it does not contain more than *k* tokens in all reachable markings, including the initial marking. It is said to be ***safe*** if it is 1-bounded; it is ***bounded*** if it is k-bounded for some *k*.

A (marked) Petri net is called *k*-bounded, *safe*, or ***bounded*** when all of its places are.

A Petri net (graph) is called ***(structurally) bounded*** if it is bounded for every possible initial marking.

Note that a Petri net is bounded if and only if its reachability graph is finite.

Soundness and Soundness Theorem

Theorem

Let PN=(P,T,F) be a workflow net, and $t^* \notin T$. PN is **sound** if and only if (PN', i), such that PN'=(P', T', F'), $T' = T \cup \{t^*\}$, and $F' = F \cup \{(o, t^*), (t^*, i)\}$, is **live** and **bounded**.

For arbitrary Petri Nets, liveness and boundedness are still complex… expect exponential run-time behaviour…

… but if we restrict to **free-choice nets,** then liveness and boundedness are computable in polynomial time!!!

Free Choice Nets

A Petri Net (P,T,F) is a free choice net if and only if for $t_1, t_2 \in T$ either $\cdot t_1 = \cdot t_2$ or $\cdot t_1 \cap \cdot t_2 = \emptyset$.

In non-free choice nets the behaviour depends on the ordering of the firing transitions…

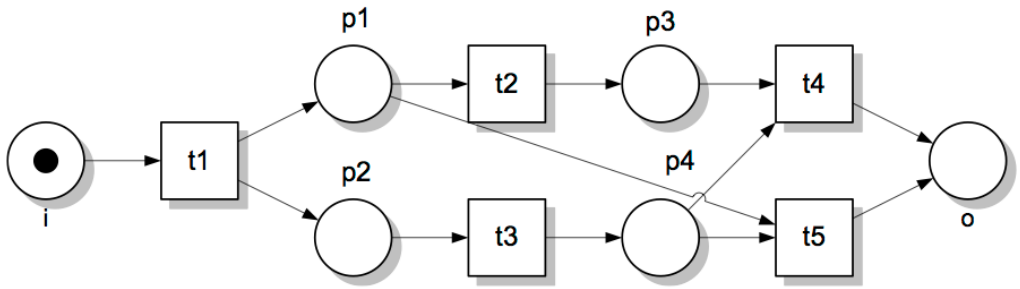


Fig. 6.14. Non-free-choice workflow net

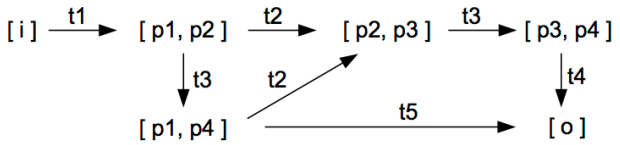


Fig. 6.15. Reachability graph of non-free-choice workflow net in Figure 6.14, showing its soundness

In this example, which transition between t4 and t5 will trigger?

The behaviour of t4, t5 depends on t2, t3 and their relative triggering order

Notice however that the Petri Net is sound!!!

Relaxed Soundness

- Soundness is a very strong property…
- … quite often, BP models derived from real industrial cases do not fit with soundness…

- The idea behind relaxed soundness is that process models are acceptable if they allow process instances with the desired properties…
- … and undesired process instances are not disallowed (while soundness doesn't permit them!)

- Relaxed Soundness: for each transition there is (at least) a firing sequence that contains it, and that finally leads to a desired process instance.

Relaxed Soundness – Definition

Definition: Let S=(PN,i) be a workflow system. Let σ, σ' be firing sequences and M, M' be states. σ is a **sound firing sequence** if it leads to a state from which a continuation to the final state [o] is possible:

$[i] \xrightarrow{\sigma} M \;\; and \;\; \exists \sigma^i \; s.t. \;\; M \xrightarrow{\sigma'} [o].$

A workflow system S=(PN, i) is **relaxed sound** if and only if each transition of PN is an element of some sound firing sequence:

$$\forall t \in T \; \exists M, M' \; : \quad \left([i] \xrightarrow{*} M \xrightarrow{t} M' \xrightarrow{*} [o] \right)$$

Weak Soundness

What happens if we use workflow nets to describe choreographies?

- Let us suppose to have a number of webservices
- The final behavior is given by combining two or more of these web services into a so-called ***choreography***
- The logic of each web service is represented through a workflow net
- Web services communicated through message passing… let us add special places representing incoming and outgoing messages
- Two services can be composed if there is at least **syntactical equivalence** between the two

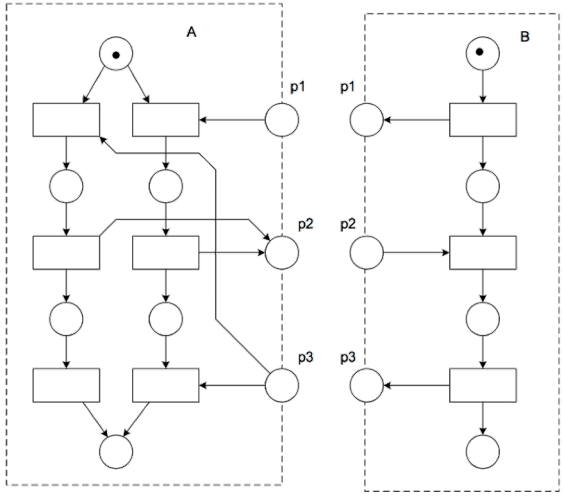


Fig. 6.23. Example of workflow modules with merged communication places

Weak Soundness

Once we have composed a Workflow net representing the whole choreography-based system, what about it?

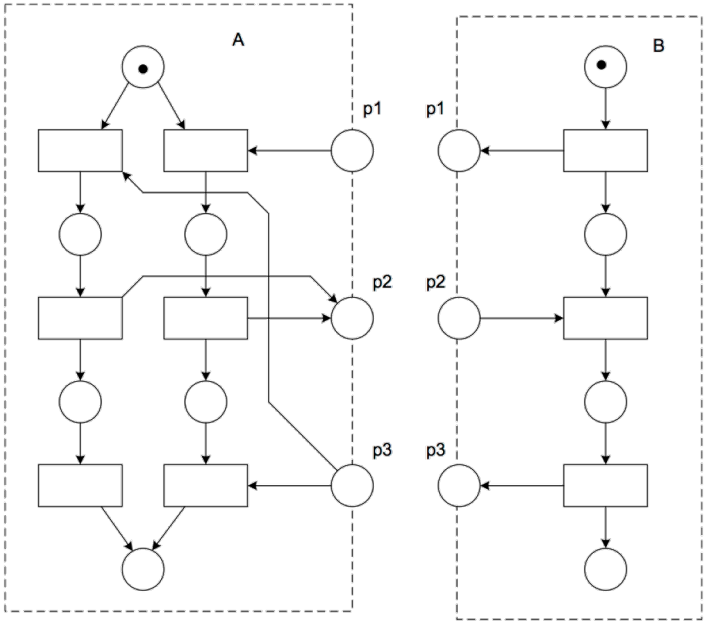


Fig. 6.23. Example of workflow modules with merged communication places

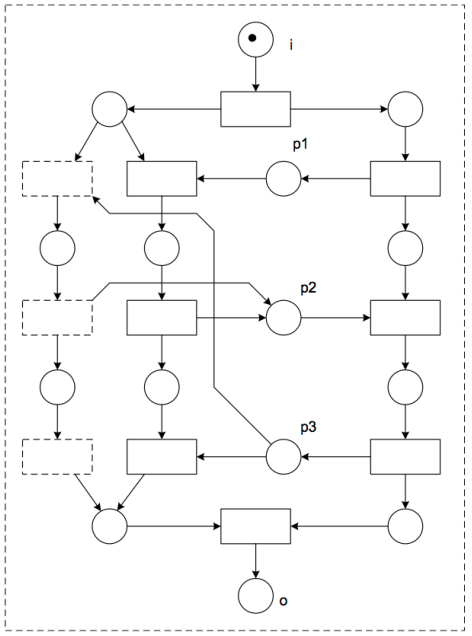


Fig. 6.24. Example of weak sound workflow net

A problem arises because, due to the composition, some paths/transition will be never enabled in the composed system… the resulting workflow would not be sound, nor relaxed sound…

Weak Soundness

A workflow system (PN, i) is **weak sound** if and only if the following holds:

- For every state M reachable from state [i] there exists a firing sequence leading from M to [o], that is:

$$\forall M \ \left([i] \xrightarrow{*} M\right) \Longrightarrow \left(M \xrightarrow{*} [o]\right)$$

- State [o] is the only state reachable from state [i] with at least one token in place o, that is:

$$\forall M \ \left([i] \xrightarrow{*} M \ \wedge M \geq [o]\right) \Longrightarrow M = [o]$$

Soundness – short recap

- **Soundness** is too restrictive, real systems do not fit very well…
- **Relaxed soundness** allows models with certain "wrong" parts, such as deadlocks… (but each activity must take part to at least one good process instance…)
- **Weak soundness** does not allow deadlocks, but it permits that certain parts will never take part to any process instance
- However, patterns discriminator, N-out-of-M join, and multiple instances Sync are not covered by any of the afore mentioned soundness criteria…

Lazy Soundness

- It allows activities to be executed after the final state has been reached
- Deadlocks are not permitted before the final state has been reached

- Activities running after the reaching of the final state are named *lazy*

Lazy Soundness – Definition

Let (PN, i) be a workflow system, with a workflow net PN=(P,T,F).
(PN, i) is lazy sound if and only if:

- $\forall M \ \left([i] \xrightarrow{*} M\right) \ \exists M' \ : \ M \xrightarrow{*} M' \ \wedge \ M'(o) = 1$
- $\forall M \ \left([i] \xrightarrow{*} M\right) \ : \ M(o) \leq 1$

In other words:

- For each state reachable from [i], there is a sequence that will lead me to the end (but other token might remain around… lazy!!!)
- No state is reachable that will have more than one token in the final place o.

Soundness criteria in other words

P1: Termination: any process instance starting from [i] will finally terminate into [o]

$$\forall M \ \left([i] \xrightarrow{*} M\right) \Longrightarrow \left(M \xrightarrow{*} [o]\right)$$

P2: Proper termination: [o] is the only state reachable from [i], with a token in the final place

$$\forall M \ \left([i] \xrightarrow{*} M \ \wedge M \geq [o]\right) \Longrightarrow M = [o]$$

P3: No dead transitions: each transition contribute to at least one process instance

$$(\forall t \in T)\ \exists M, M' \ : \ [i] \xrightarrow{*} M \xrightarrow{t} M'$$

P4: Transition Participation: each transition participates to at least a process instance that starts from the initial state and terminates in the final state

$$(\forall t \in T)\ \exists M, M' \ : \left([i] \xrightarrow{*} M \xrightarrow{t} M' \xrightarrow{*} [o]\right)$$

Soundness criteria in other words

Noteworthy results:

Soundness $\Leftrightarrow \boldsymbol{P1} \wedge \boldsymbol{P2} \wedge \boldsymbol{P3}$

Weak Soundness $\Leftrightarrow \boldsymbol{P1} \wedge \boldsymbol{P2}$

Relaxed Soundness $\Leftrightarrow \boldsymbol{P4}$

Why formal properties? Which formal properties?

- If a process model guarantees a property, all process instances will ensure that property
- It is a way to ensure that **by design** our process instance will exhibit a certain behaviour

- Two different types of properties:
  - Structural, control-flow related properties
  - **Domain related properties ?????????**

- Will my process ever reach a certain outcome?
- Under which circumstance will my process reach a certain outcome?
- Is it the case that some path will be never walked down?
- Will my process ever end?
- Will always be the case that if *XXX*, then also *YYY* ?

LTL and BPM:
Linear-time Temporal Logic and its use in the Business Process Modelling

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

(propositional) Liner-time Temporal Logic

LTL is a **modal logic** built on **atomic propositions**, whose truthiness can change along time.

It is based on the notion of a ***world***, described in terms of propositions that are true in that world, and on a function that make each world to evolve into a new one.

Intuitively, the evolution from one world *w* to the subsequent one *w'* is exploited to represent the passing of time.

Sources:

Emerson EA. Temporal and Modal Logic. In: van Leeuwen J (ed.), Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics. Elsevier and MIT Press. 1990. ISBN: 0-444-88074-7, 0-262-22039-3.

(propositional) Liner-time Temporal Logic

Generally speaking, LTL denotes a temporal logic that is:

- **linear**: each situation has a single next future moment (world), and **temporal modalities (temporal operators)** therefore predicate on the truth of propositions along a single timeline;
- **qualitative**: Temporal operators are used to express qualitative time relations between propositions. Notice that metric distances are not part of the logic…;
- **point-based**: operators and propositions are evaluated over points in time, i.e. over the worlds that evolve;
- **discrete**: the present moment corresponds to the current state of the system and the next one to its immediate successor (where the first future event happens);
- **future-tense**: temporal operators refer to the occurrence of events in the future.

Axioms for modal logics in general

Modal logics can be characterized in general by their axioms. Some well-known and important axioms are the following (picture taken from Modality by Paul Portner, Oxford University Press, 2009)

Axiom name	Axiom
K	$\Box(p \rightarrow q) \rightarrow (\Box p \rightarrow \Box q)$
T	$\Box p \rightarrow p$
B	$p \rightarrow \Box\Diamond p$
D	$\Box p \rightarrow \Diamond p$
4	$\Box p \rightarrow \Box\Box p$
E	$\Diamond p \rightarrow \Box\Diamond p$

System name	Axioms
K	K
T	K T
B	K T B
D	K D
S4	K T 4
S5	K T E

LTL Model – Definition

- Formally, an LTL **time structure F**, also called **frame**, models a single linear timeline. F is a totally ordered set (K,<).
- In the following K will be assumed to be the set of natural numbers.

Definition 2.1. (LTL model)
Let $\mathcal{P}$ be the set of all atomic propositions in the system. An LTL model $\mathcal{M}$ for $\mathcal{P}$ is a triple $(\mathcal{K}, \prec, \mathrm{v})$ where $\mathrm{v} : \mathcal{P} \rightarrow 2^{\mathcal{K}}$ is a *function* which maps each proposition in $\mathcal{P}$ to the set of time instants at which the proposition holds.

LTL Execution Trace – Definition

We are interested in mapping business processes, and in particular to reasons about events happening in the system…

- … the idea is that the state of a system at each time instant can be described by the events happening at that time instant
- … and that the evolution of a system is given by the sequences of system states, i.e. by the sequence of happened events

Definition 2.2. (LTL execution trace)
Given a set $\mathcal{E}$ of atomic propositions (representing possible events), an *LTL execution trace* $\mathcal{T}_{\text{LTL}}$ is an LTL model having $(\mathbb{N}, <)$ as time structure and $\mathcal{E}$ as the set of atomic propositions. In particular, $\mathcal{T}_{\text{LTL}} = (\mathbb{N}, <, v_{occ})$, where $v_{occ} : \mathcal{E} \rightarrow 2^{\mathbb{N}}$ is a valuation function mapping each event $e \in \mathcal{E}$ to the set of all time instants $i \in \mathbb{N}$ at which e occurs.

LTL Execution Trace – Differences w.r.t. classic LTL

Definition 2.2. (LTL execution trace)
Given a set $\mathcal{E}$ of atomic propositions (representing possible events), an *LTL execution trace* $\mathcal{T}_{\text{LTL}}$ is an LTL model having $(\mathbb{N}, <)$ as time structure and $\mathcal{E}$ as the set of atomic propositions. In particular, $\mathcal{T}_{\text{LTL}} = (\mathbb{N}, <, v_{occ})$, where $v_{occ} : \mathcal{E} \to 2^{\mathbb{N}}$ is a valuation function mapping each event $e \in \mathcal{E}$ to the set of all time instants $i \in \mathbb{N}$ at which e occurs.

1. Trace length: classic LTL models are used for reasoning and represent infinite traces. Clearly, in BPM such assumption is wrong, since BPM traces must be finite.
   Consequence: classical LTL semantics over infinite trace need to be adjusted for finite traces (n turn, a number of further consequences).

2. Can event happens simultaneously? In classical LTL, yes, they do!; when using them in BPM, no…
   Assumption: at each time instant at most one event can happen; further assumption: at each time instant at least one event can happen.

Syntax of LTL Formulae

LTL formulae are made of three ingredients:

1. Atomic propositions: the events that are happening in a certain state, plus two special constants *true* and *false*.
2. Classical Logical propositional connectives:¬, ∧, ∨ and ⟹
3. Temporal operators:

   ◯ (next time)

   $\mathcal{U}$ (until)

   ◊ (eventually)

   □ (globally)

   $\mathcal{W}$ (weak until)

Syntax of LTL Formulae – other temporal operators

LTL formulae are made of three ingredients:

1. Atomic propositions: the events that are happening in a certain state, plus two special constants *true* and *false*.
2. Classical Logical propositional connectives:¬, ∧, ∨ and $\Longrightarrow$
3. Other temporal operators:

- $\varphi \vee \psi \triangleq \neg(\neg\varphi \wedge \neg\psi)$ and $\varphi \Rightarrow \psi \triangleq \neg\varphi \vee \psi$;
- $true \triangleq \neg\varphi \vee \varphi$ and $false \triangleq \neg true$;
- $\Diamond\varphi \triangleq true\mathcal{U}\varphi$;
- $\Box\varphi \triangleq \neg\Diamond\neg\varphi$;
- $\psi\mathcal{W}\varphi \triangleq \psi\mathcal{U}\varphi \vee \Box\psi$.

LTL Semantics (1/2)

The semantics of LTL is given with respect to an LTL execution trace, and with respect to a specific state. We will use $\models_{\text{LTL}}$ to denote the logical *entailment* in the LTL setting. $\mathcal{M}, i \models_{\text{LTL}} \varphi$ means that φ is true at time i in model $\mathcal{M}$. $\models_{\text{LTL}}$is defined by induction on the structure of the formulae[2]:

$(\mathcal{T}_{\text{LTL}} \models_{\text{LTL}} \varphi)$ iff $(\mathcal{T}_{\text{LTL}}, 0 \models_{\text{LTL}} \varphi)$;

$(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} e)$ iff $e \in \mathcal{T}_{\text{LTL}}(i)$ (i.e., $i \in v_{occ}(e)$);

$(\mathcal{T}_{\text{LTL}}, i \not\models_{\text{LTL}} e)$ iff $e \notin \mathcal{T}_{\text{LTL}}(i)$;

$(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \neg\varphi)$ iff $(\mathcal{T}_{\text{LTL}}, i \not\models_{\text{LTL}} \varphi)$;

$(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \varphi \wedge \psi)$ iff $(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \varphi)$ and $(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \psi)$;

$(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \varphi \vee \psi)$ iff $(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \varphi)$ or $(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \psi)$;

LTL Semantics (2/2)

The semantics of LTL is given with respect to an LTL execution trace, and with respect to a specific state. We will use $\models_{\text{LTL}}$ to denote the logical *entailment* in the LTL setting. $\mathcal{M}, i \models_{\text{LTL}} \varphi$ means that φ is true at time i in model $\mathcal{M}$. $\models_{\text{LTL}}$is defined by induction on the structure of the formulae[2]:

$(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \varphi \Rightarrow \psi)$ iff $(\mathcal{T}_{\text{LTL}}, i \not\models_{\text{LTL}} \varphi)$ or $(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \psi)$;

$(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \bigcirc\varphi)$ iff $(\mathcal{T}_{\text{LTL}}, i+1 \models_{\text{LTL}} \varphi)$;

$(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \psi\mathcal{U}\varphi)$ iff $\exists k \geq i$ s.t. $(\mathcal{T}_{\text{LTL}}, k \models_{\text{LTL}} \varphi)$ and $\forall j. i \leq j < k$ $(\mathcal{T}_{\text{LTL}}, j \models_{\text{LTL}} \psi)$;

$(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \Diamond\varphi)$ iff $\exists j \geq i$ s.t. $(\mathcal{T}_{\text{LTL}}, j \models_{\text{LTL}} \varphi)$;

$(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \Box\varphi)$ iff $\forall j \geq i$ $(\mathcal{T}_{\text{LTL}}, j \models_{\text{LTL}} \varphi)$;

$(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \psi\mathcal{W}\varphi)$ iff either $(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \psi\mathcal{U}\varphi)$ or $(\mathcal{T}_{\text{LTL}}, i \models_{\text{LTL}} \Box\psi)$.

LTL and BPM – few examples

a) In my bar, if you have a coffee then you will pay it.

b) In my bar, if you have a coffee, you will pay for it. If you pay for a coffee, then you will have a coffee.

c) In my bar, it is always true that after payment, they will tell you "goodbye"

How properties are checked in systems described through LTL?

Roughly speaking…

1. Dynamic of a system is represented as an LTL formula
2. Property to be verified is negated and represented as an LTL formula
3. Two different automata are built for each formula (Buchi automata)
4. The two automata are composed, and liveness properties are looked for… if found, the system does no entail the initial property

The above procedure falls in the Model Checking verification technique, a very broad and large way for verifying formal properties of systems.

A number of Model Checkers are available, some of them free, some of them with "industrial strength".

As an example, see the Spin Model Checker:
http://spinroot.com/spin/whatispin.html

DECLARE

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Declarative, Open Process Modeling: Declare

- Appears in 2006 in two different workshops (WS-FM2006, BPM Workshops2006), under the name DecSerFlow
- Presented as DECLARE in later works:
  Wil M. P. van der Aalst, Maja Pesic, Helen Schonenberg: Declarative workflows: Balancing between flexibility and support. Computer Science - R&D 23(2): 99-113 (2009)
  Maja Pesic, Helen Schonenberg, Wil M. P. van der Aalst: Declarative Workflow. Modern Business Process Automation 2010: 175-201
- Fully presented in Maja Pesic Ph.D. Thesis: the language is ConDec, the overall approach is DECLARE

- Marco Montali, Maja Pesic, Wil M. P. van der Aalst, Federico Chesani, Paola Mello, Sergio Storari: Declarative specification and verification of service choreographies. TWEB 4(1): 3:1-3:62 (2010)

A completely new (for the BP community) approach to flow specification

The focus is not on the flow, but on the **properties the flow should exhibit**

Based on **constraints** between activity executions

Declarative, Open Process Modeling: Declare

Based on

- declarative approach: constraints about the happening of events/ prohibition of events
- open: any activity that is not strictly forbidden, is allowed

Supports:

- Data
- Resources (through data)

- Few free editors
- Based on LTL Formal semantics: Model Checking techniques available

- Difficult to move to workflow engine
- Better for monitoring components on process executions

Declarative, Open Process Modeling: Declare

Unary Constraints:

- Atomic activities are represented as boxes
- Existence, existence_N, and absence constraints are defined as simple annotations on the boxes

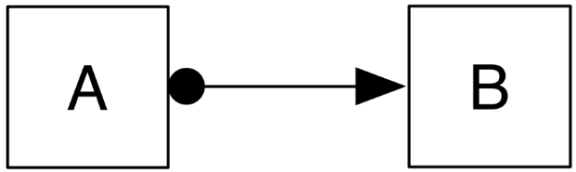
Binary Constraints:

- Connects two (or more) activities
- A solid circle indicates the event (activity execution) that triggers the constraint
- If present, an arrow signals a temporal ordering

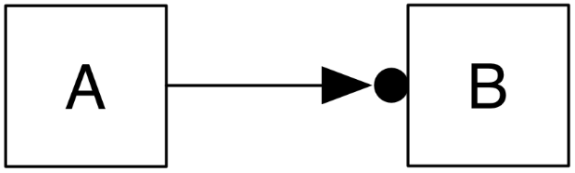
Declarative, Open Process Modeling: Declare

Binary Constraints:

- Connects two (or more) activities
- A solid circle indicates the event (activity execution) that triggers the constraint
- If present, an **arrow signals a temporal ordering**



Response



Precedence

Response: An execution of A should be always followed by an execution of B. Allowed traces:

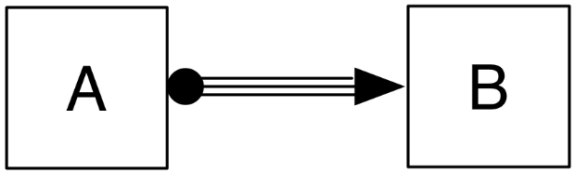
- B
- B B B
- A B B
- A A A A A A B

Declarative, Open Process Modeling: Declare

Binary Constraints:

- Connects two (or more) activities
- A solid circle indicates the event (activity execution) that triggers the constraint
- If present, **an arrow signals a temporal ordering**
- **Two lines** in the connection stand for no repetition of the triggering activity before the constraint is satisfied
- **Three lines** in the connection stand for no other activity is allowed before the constraint is satisfied…

Chained Response



Allowed traces:

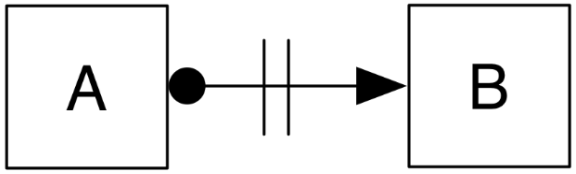
- B
- A B A B A B
- C C C A B B B A B C C C

Declarative, Open Process Modeling: Declare

Binary Constraints:

- Connects two (or more) activities
- A solid circle indicates the event (activity execution) that triggers the constraint
- If present, an arrow signals a temporal ordering
- **Two lines** in the connection stand for no repetition of the triggering activity before the constraint is satisfied
- **Three lines** in the connection stand for no other activity is allowed before the constraint is satisfied…
- constraints might be negated:

Negated Response



Intended meaning: after A, B cannot be executed

Declare Semantics

Table V. Some DecSerFlow Templates

Name	LTL Expression	Graphical
existence(*A*)	$\Diamond(A)$	1..* A
existence_2(*A*)	$\Diamond(A \wedge \bigcirc(existence(A)))$	2..* A
existence_3(*A*)	$\Diamond(A \wedge \bigcirc(existence2(A)))$	3..* A
absence_2(*A*)	$\neg existence2(A)$	0..1 A
absence_3(*A*)	$\neg existence3(A)$	0..2 A
exactly_1(*A*)	$existence(A) \wedge absence2(A)$	1 A
exactly_2(*A*)	$existence2(A) \wedge absence3(A)$	2 A
response(*A*, *B*)	$\Box(A \Rightarrow \Diamond(B))$	A → B
precedence(*A*, *B*)	$\Diamond(B) \Rightarrow ((\neg B) \sqcup A)$	A ⇢ B
succession(*A*, *B*)	$response(A, B) \wedge precedence(A, B)$	A ⇢ B
neg_response(*A*, *B*)	$\Box(A \Rightarrow \neg(\Diamond(B)))$	A ⇸ B
responded_existence(*A*, *B*)	$(\Diamond A) \Rightarrow (\Diamond B)$	A — B
alternate_response(*A*, *B*)	$response(A, B) \wedge \Box(A \Rightarrow \bigcirc(precedence(B, A))$	A ⟹ B
chain_response(*A*, *B*)	$\Box(A \Rightarrow \bigcirc(B))$	A ⟹ B

Declare – an example

- if the warehouse cannot ship the order, then the seller must refuse it;
- the seller can accept the order only if the warehouse has previously accepted its shipment;
- both the seller and the warehouse cannot accept and reject the same order, that is, answers are mutually exclusive.

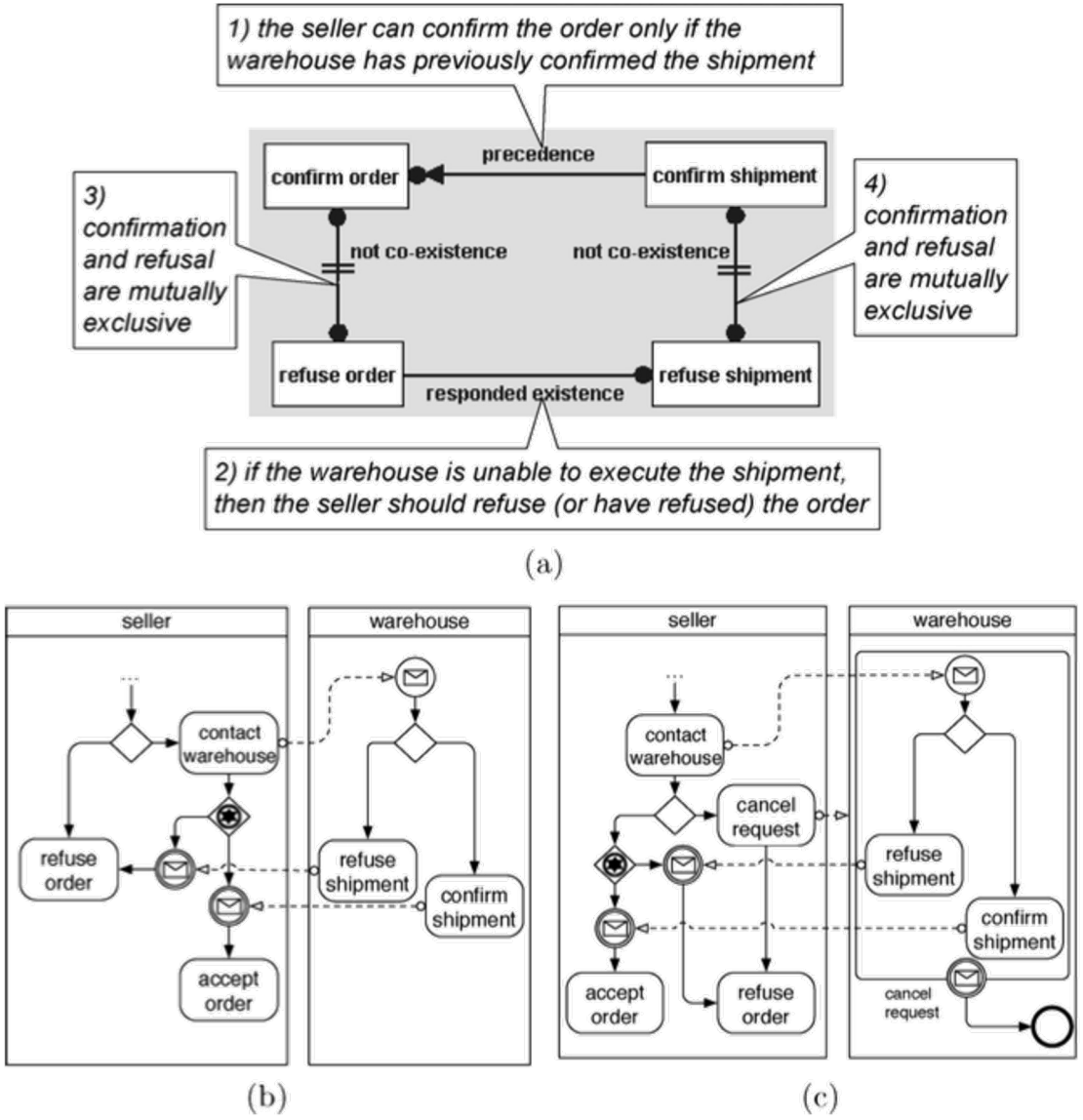
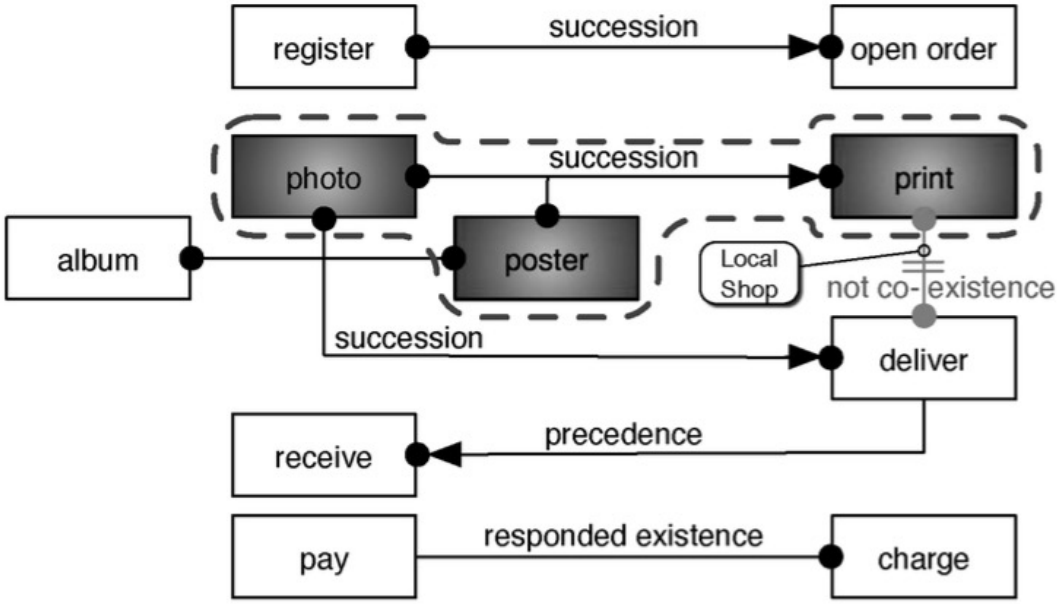


Fig. 1. Declarative vs. procedural style of modeling a simple choreography.

Declare – which properties can be verified?

Enactment of a process model

- At the beginning of the process instance execution, which are the activities that can be executed?
- At any moment in the process instance execution, which are the activities that can be executed next?
- At any moment in the process instance execution, is there a way to satisfy all the constraints?



(a) Dead activities in a DecSerFlow composition ("photo", "poster" and "print").

Declare – which properties can be verified?

Conformance checking

- Given a log of an executed process instance…
- …does the log respect all the constraints?
- in constraints programming terminology, is the trace a solution of the constraint program?

- If not, which constraints are violated? Many issues on this aspect…

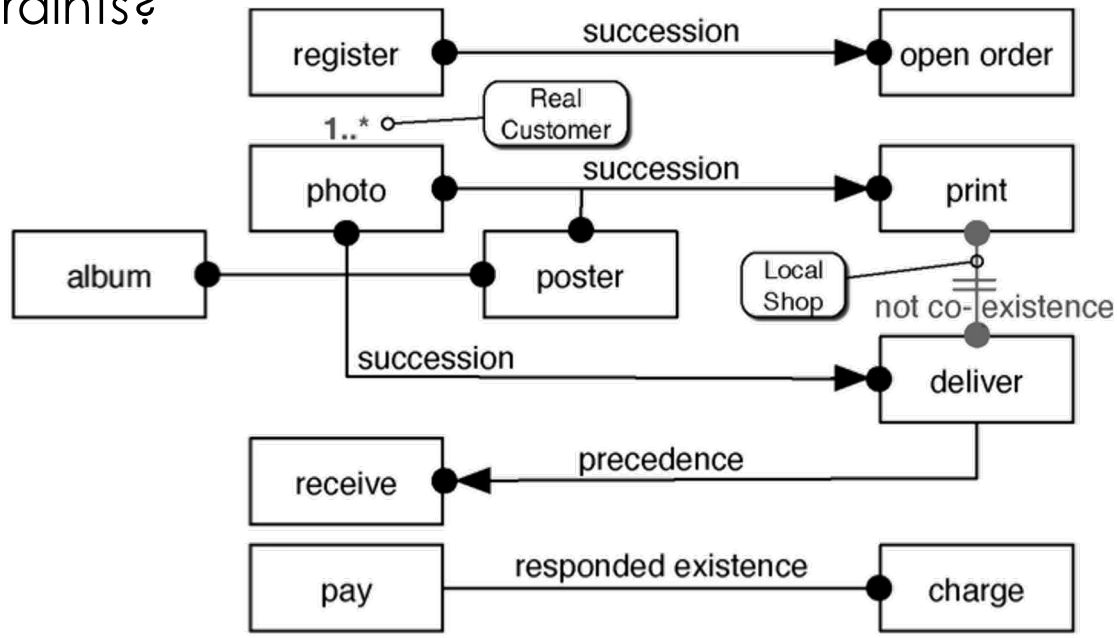
Declare – which properties can be verified?

Interoperability

- Two different systems are represented through DECLARE models…
- Are they interoperable?

- Let us suppose to join the two Declare models, by adding some constraints on the "interfacing" activities

- Is there still a way to execute the process instance, and satisfy all the constraints?



(b) Conflict in a DecSerFlow composition.

Declare – Limits

- **What about quantitative temporal constraints**?
  LTL logic is propositional… no way for Declare to support quantitative temporal constraints.
  Need to equip Declare with a different semantics, as it has been done in Montali, Pesic, van der Aalst, Chesani, Mello, Storari: Declarative specification and verification of service choreographies. TWEB 4(1): 3:1-3:62 (2010)
  Semantics given in terms of the SCIFF framework, Abductive Logic Programming
  However, decidability issues might arise…

- **What about data, data constraints, data object life cycles?**
  Same problem with limits of the LTL semantics.
  A solution was proposed in Montali, Maggi, Chesani, Mello, van der Aalst, Monitoring business constraints with the event calculus. ACM TIST 5(1): 17:1-17:30 (2013)
  Semantics given in terms of Event Calculus

Declare – Limits

- **What about automatic execution?**
  The constraint based approach provides a greater flexibility…
  … i.e., it allows a certain degree of indeterminism…
  … no way it can be automatically executed …

- **What about learning models?**
  Best learning algorithms for process models work with procedural, closed languages as target.
  A solution was proposed in Lamma, Mello, Montali, Riguzzi, Storari: Inducing Declarative Logic-Based Models from Labeled Traces. BPM 2007: 344-359
  Use of Inductive Logic Programming learning technique, but needs positive and negative labelled traces

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Federico Chesani

DISI – Department of Computer Science and Engineering

federico.chesani@unibo.it

www.unibo.it