



***Lab for the course on Process and Service Modeling and Analysis***

# **LAB-07**

## **Declarative Process Modeling and Mining**

**Lecturer: Andrea MARRELLA**

*Courtesy of Fabrizio Maria Maggi and Claudio Di Ciccio*



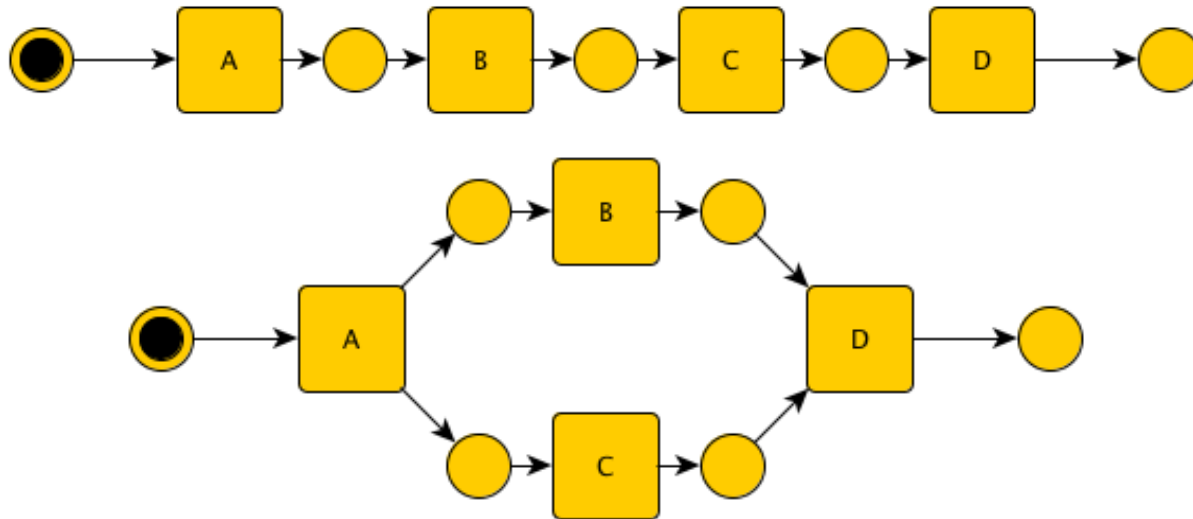
# Outline

- Imperative vs Declarative process modelling
- Declarative process modeling with **DECLARE**
- Instantiation of declarative constraints through **DECLARE templates**
- Declarative Process Mining in ProM
- Classroom Exercises



# Imperative Process Models

- An imperative process model represents the whole process behaviour **at once**.
  - The most used notation is based on a subclass of Petri Nets (namely, the **Workflow Nets**). Other extension exist (e.g., **BPMN**).

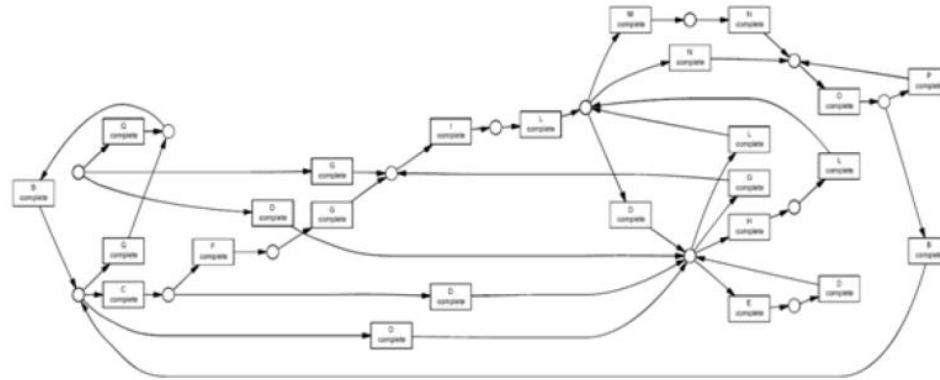


Imperative process models *explicitly specify* all possible behaviors (closed models).



# Imperative Process Models in Stable Environments

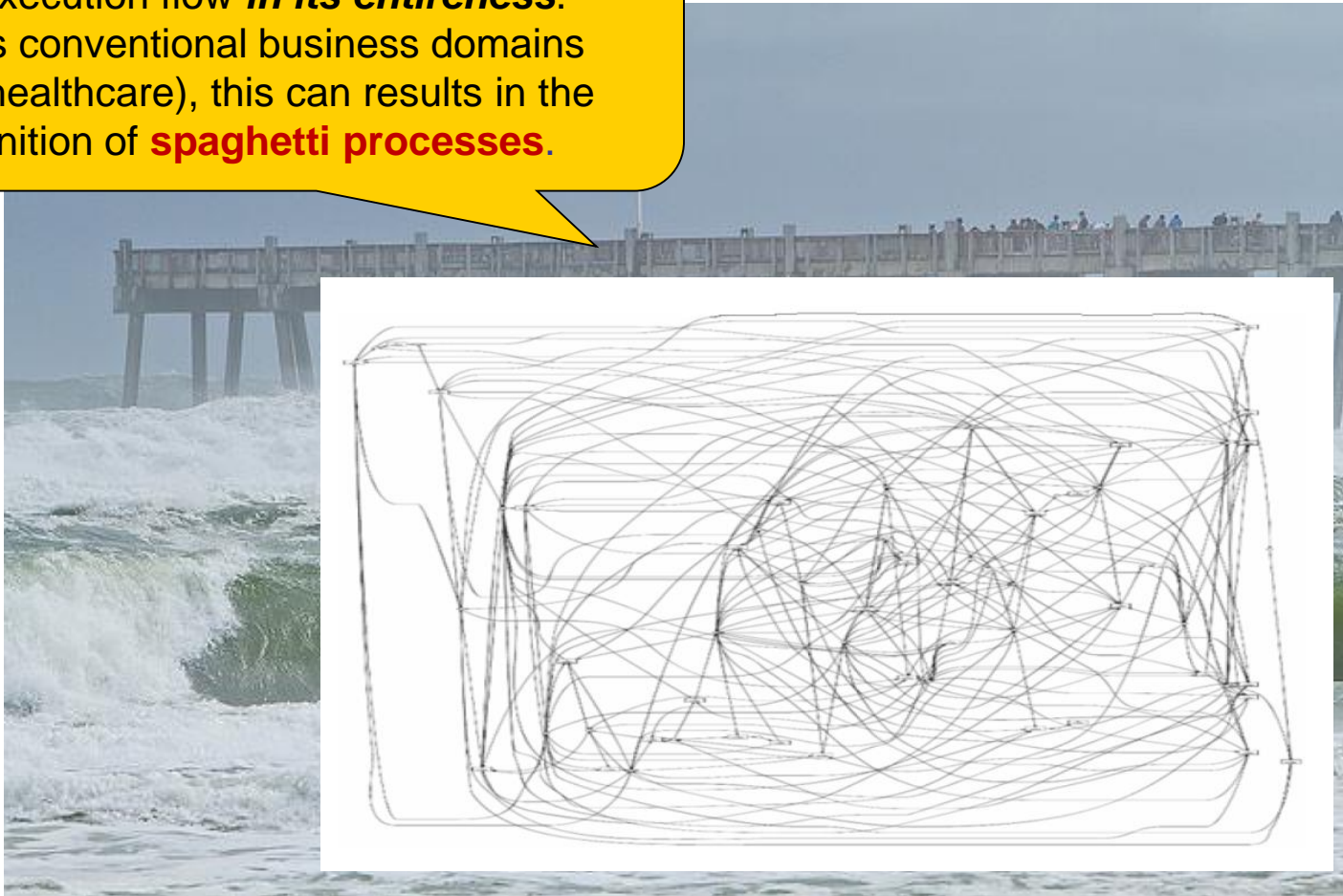
Imperative process models represent well the behaviour of processes in **stable business-oriented environments**. This kind of structured work includes mainly production and administrative processes.





# Imperative Process Models in Turbulent Environments

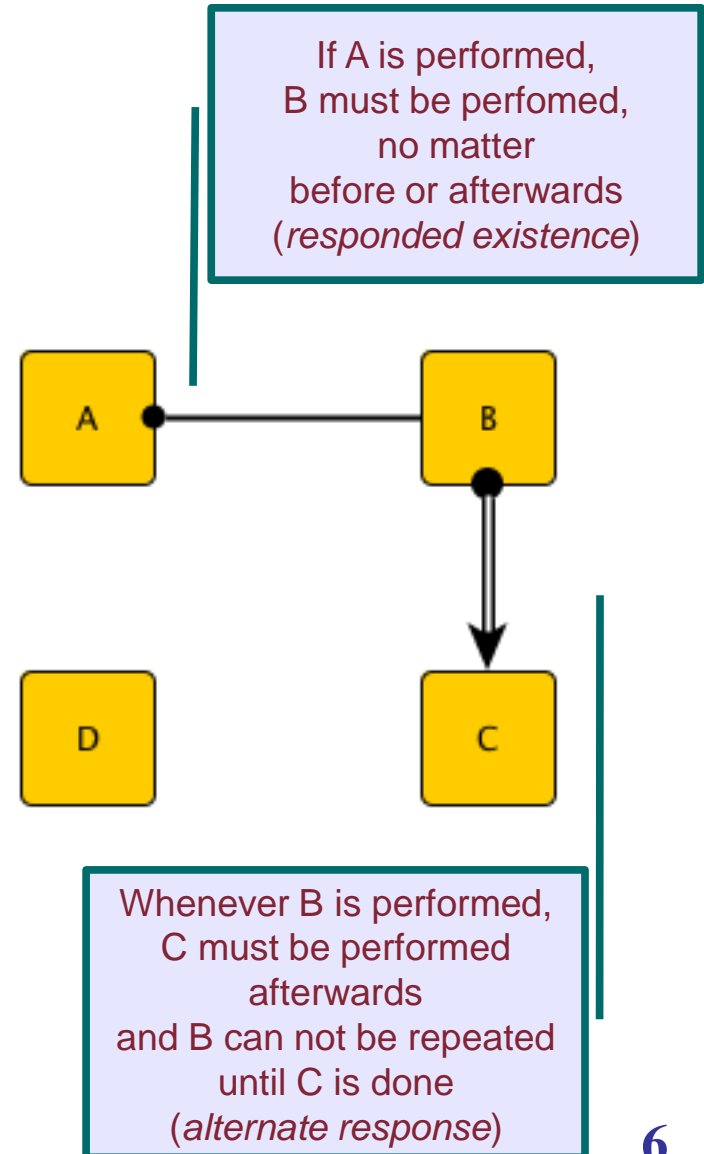
Imperative process models prescribe the execution flow *in its entirety*. In less conventional business domains (e.g., healthcare), this can result in the definition of **spaghetti processes**.





# Declarative Process Models

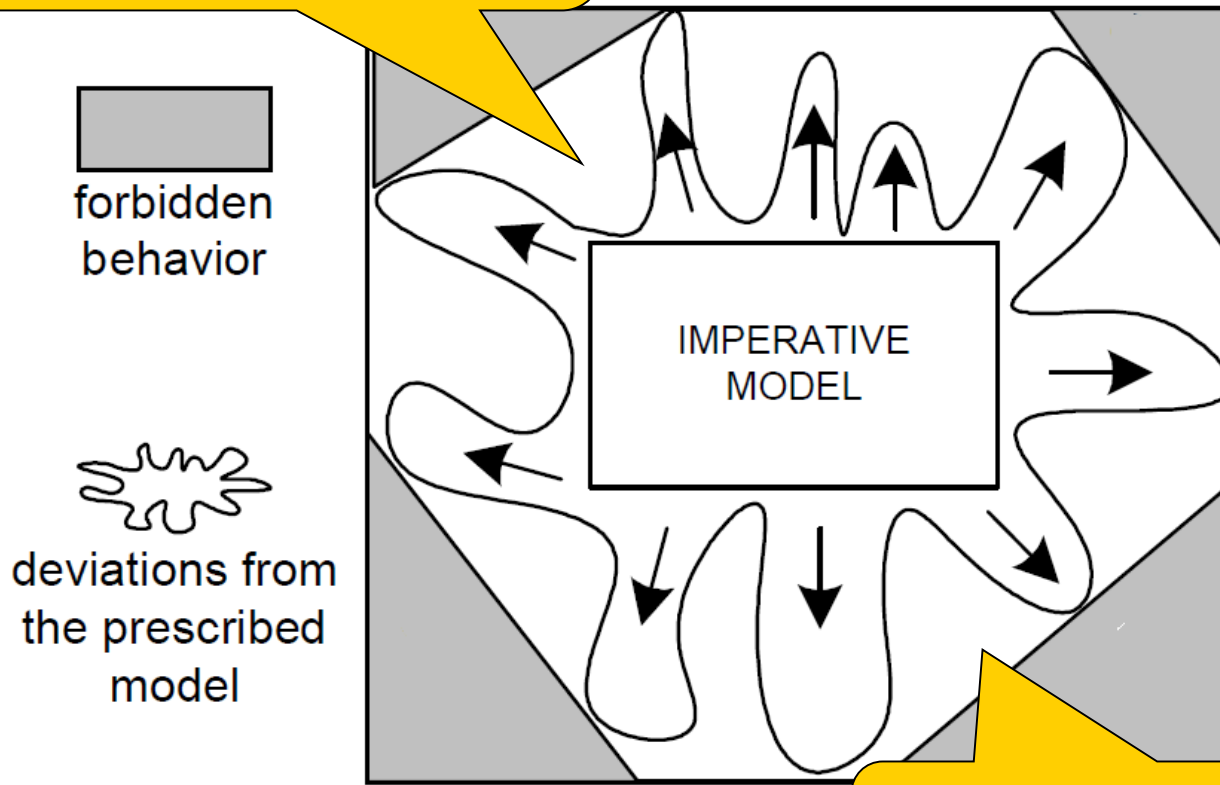
- Rather than using an imperative language for expressing the allowed sequence of activities, **declarative process models** are based on the description of business processes through the usage of **constraints**.
- Such constraints **implicitly specify** the **allowed behaviour** of the process.
- The idea is that **every task can be performed, except the ones** which do not respect such constraints.
- Declarative models are appropriate to describe **dynamic environments**, where processes are highly flexible and subject to changes.





# Imperative vs Declarative Models

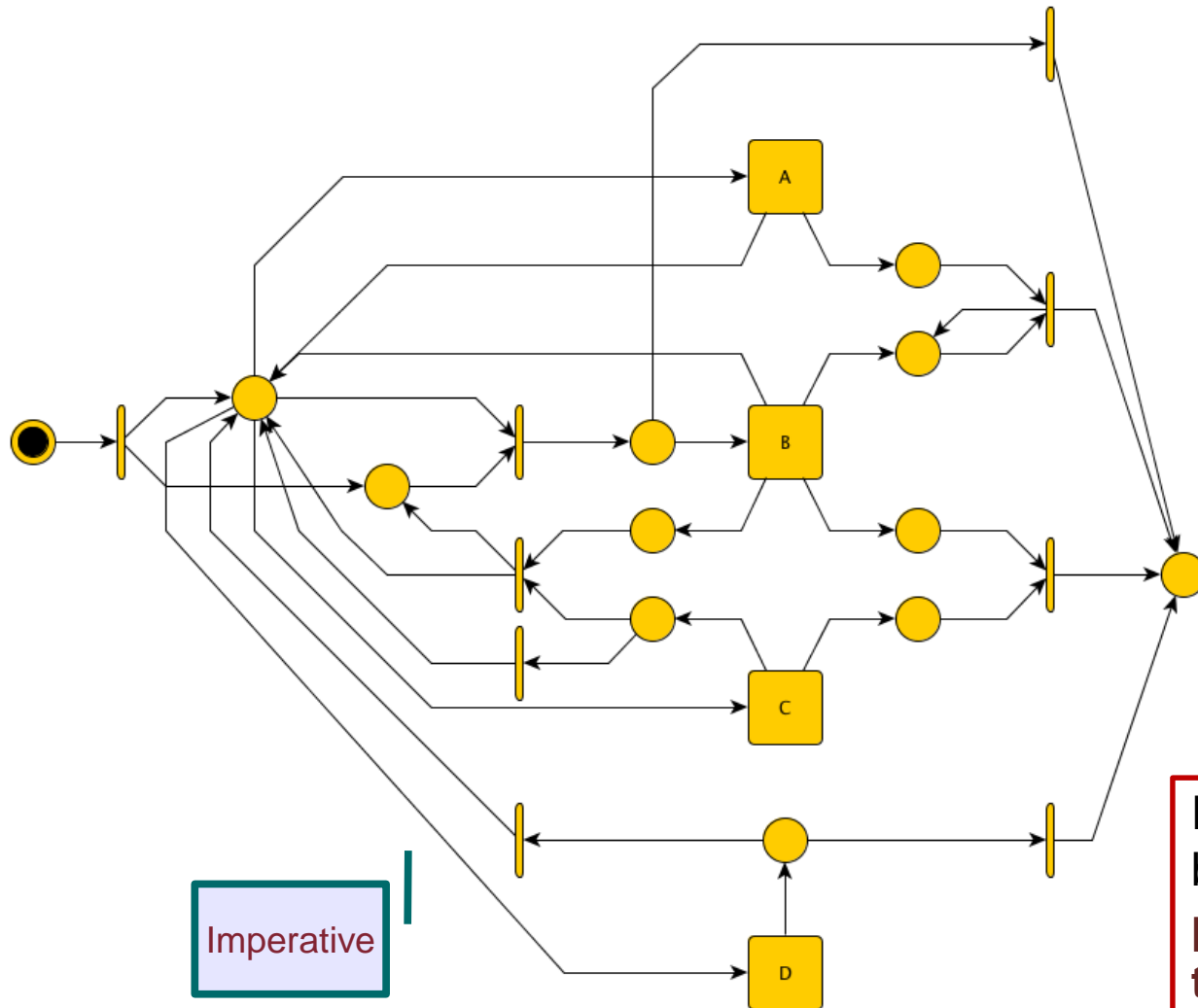
Imperative process models **explicitly specify** all possible sequences of activities in a process.



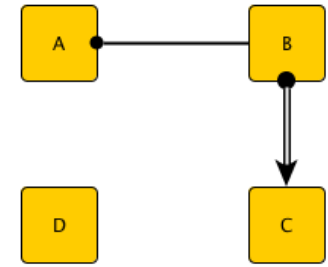
Declarative process models offer more flexibility: **everything that is not specified is allowed.**



# Imperative vs Declarative Models



Imperative



Declarative

Declarative models work better in presence of a **partial specification of the process scheme.**





# The DECLARE Process Modeling Language

- DECLARE is a declarative process modeling language originally introduced in:

Wil MP van Der Aalst, Maja Pesic, Helen Schonenberg

***Declarative workflows: Balancing between flexibility and support***

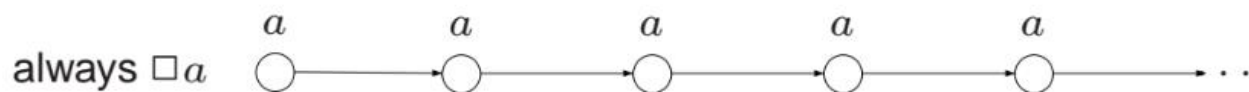
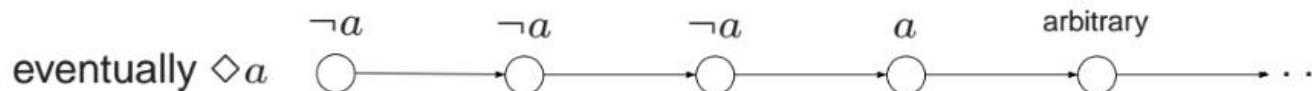
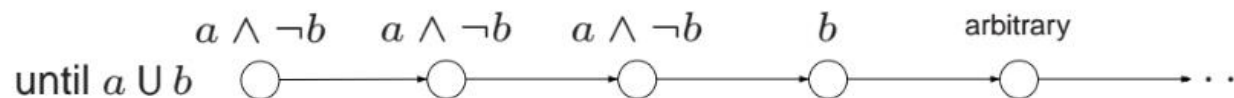
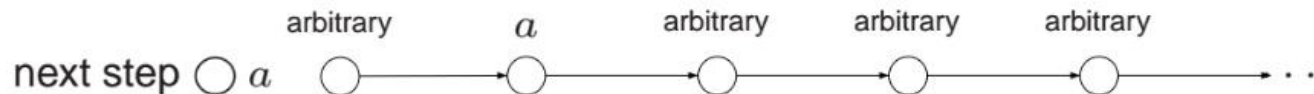
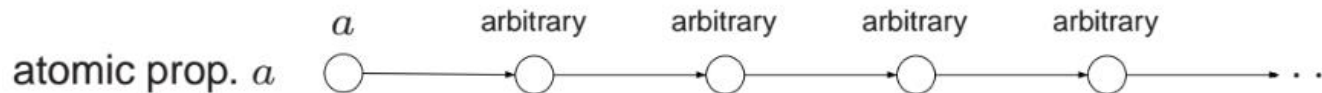
Computer Science-Research and Development vol.23, n.2 (2009)

- Technically a DECLARE model  $D = (A, \pi_D)$  consists of a set of possible activities  $A$  involved in a process and a collection of ***temporal constraints***  $\pi_D$  defined over such activities.
- DECLARE constraints are instantiation of ***templates***, i.e., patterns that define parameterized classes of properties.
- Templates have a ***graphical representation*** and enjoy a precise semantics in ***LTL over finite traces***.



# Recap: LTL Operator Semantics

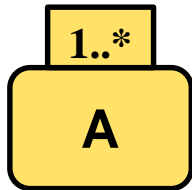
<i>operator</i>	<i>semantics</i>
$\bigcirc\varphi$	$\varphi$ has to hold in the next position of a path.
$\square\varphi$	$\varphi$ has to hold always in the subsequent positions of a path.
$\diamond\varphi$	$\varphi$ has to hold eventually (somewhere) in the subsequent positions of a path.
$\varphi U \psi$	$\varphi$ has to hold in a path at least until $\psi$ holds. $\psi$ must hold in the current or in a future position.
$\varphi W \psi$	$\varphi$ has to hold in the subsequent positions of the log at least until $\psi$ holds. If $\psi$ never holds, $\varphi$ must hold everywhere.





# DECLARE constraint templates

## Existence templates



### Existence(A)

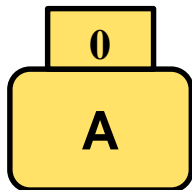
LTL Formalization:  $\diamond A$

Activity *A* occurs at least 1 time in the process instance.

*BCAAC* ✓

*BCAAAC* ✓

*BCC* ✗



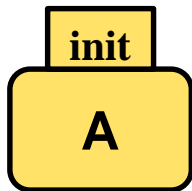
### Absence(A)

LTL Formalization:  $\neg \diamond A$

Activity *A* does not occur in the process instance.

*BCC* ✓

*BCAC* ✗



### Init(A)

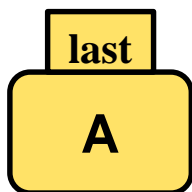
LTL Formalization:  $A$

Activity *A* is the first to occur in each process instance.

*BCAAC* ✗

*ACAAAC* ✓

*BCC* ✗



### Last(A)

LTL Formalization:  $\diamond(A \wedge O \neg T)$

Activity *A* is the last to occur in each process instance.

*BCAAC* ✗

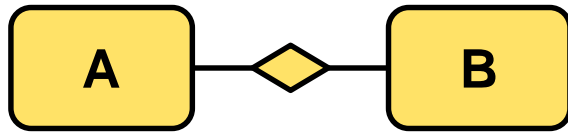
*ACAAAC* ✗

*BCA* ✓



# DECLARE constraint templates

## Choice templates

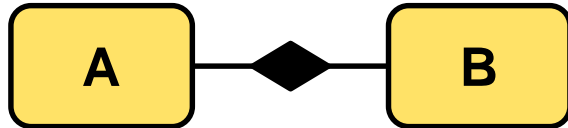


### Choice(A,B)

LTL Formalization:  $\diamond A \vee \diamond B$

Activity *A* or *B* eventually occur in the process instance.

*BCAAC* ✓ *CDC* ✗ *BCC* ✓



### Exclusive Choice(A,B)

LTL Formalization:  $(\diamond A \vee \diamond B) \wedge \neg(\diamond A \wedge \diamond B)$

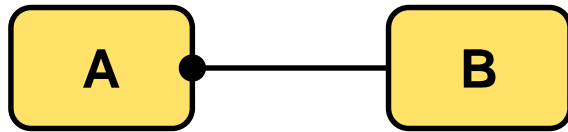
Activity *A* or *B* eventually occur in the process instance, but not together.

*BCAAC* ✗ *CDC* ✗ *BCC* ✓



# DECLARE constraint templates

## Relation templates

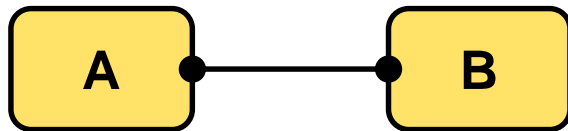


### RespondedExistence(A, B)

LTL Formalization:  $\diamond A \rightarrow \diamond B$

If *A* occurs in the process instance, then *B* occurs as well.

*CAC* ✗ *CAACB* ✓ *BCAC* ✓ *BCC* ✓



### Co-Existence

LTL Formalization:  $(\diamond A \rightarrow \diamond B) \wedge (\diamond B \rightarrow \diamond A)$

Activity *A* or *B* eventually occur in the process instance.

*BCAAC* ✓ *CDC* ✗ *BCC* ✗



# DECLARE constraint templates

## Relation templates



### Response(A, B)

LTL Formalization:  $\square (A \rightarrow \diamond B)$

If *A* occurs in the process instance, then *B* occurs after *A*.  
*BCAAC* ✗ *CAACB* ✓ *CAC* ✗ *BCC* ✓



### AlternateResponse(A, B)

LTL Formalization:  $\square (A \rightarrow O(\neg A \cup B))$

Each time *A* occurs in the process instance, then *B* occurs afterwards, before *A* recurs. *BCAAC* ✗ *CAACB* ✗  
*CACB* ✓ *CABCA* ✗ *BCC* ✓ *CACBBAB* ✓



### ChainResponse(A, B)

LTL Formalization:  $\square (A \rightarrow OB)$

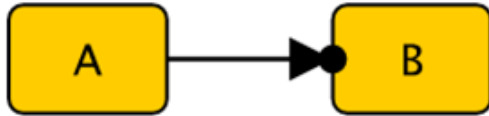
Each time *A* occurs in the process instance, then *B* occurs immediately afterwards.

*BCAAC* ✗ *BCAABC* ✗ *BCABABC* ✓



# DECLARE constraint templates

## Relation templates

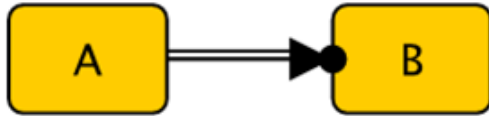


### Precedence(A,B)

LTL Formalization:  $\neg \mathbf{B} \mathbf{W} \mathbf{A}$

$B$  occurs in the process instance only if preceded by  $A$

$BCAAC$  ✗  $CAACB$  ✓  $CAC$  ✓



### AlternatePrecedence(A,B)

LTL Formalization:  $(\neg \mathbf{B} \mathbf{W} \mathbf{A}) \wedge \square (\mathbf{B} \rightarrow (\neg \mathbf{B} \mathbf{W} \mathbf{A}))$

Each time  $B$  occurs in the process instance, it is preceded by  $A$  and no other  $B$  can recur in between.

$BCC$  ✗  $BCAAC$  ✗  $CAACB$  ✓  $CACB$  ✓  $CABCA$  ✓  $CACBAB$  ✓



### ChainPrecedence(A,B)

LTL Formalization:  $\square (\mathbf{OB} \rightarrow \mathbf{A})$

Each time  $B$  occurs in the process instance, then  $A$  occurs immediately beforehand

$BCAAC$  ✗  $BCAABC$  ✗  $CABABCA$  ✓



# DECLARE constraint templates

## Relation templates



### Succession(A, B)

LTL Formalization:  $\Box (\mathbf{A} \rightarrow \Diamond \mathbf{B}) \wedge (\neg \mathbf{B} \mathbf{W} \mathbf{A})$

$A$  ( $B$ ) occurs if and only if it is followed (preceded) by  $B$  ( $A$ ) in the process instance

$BCAAC$  ✗  $CAACB$  ✓  $CAC$  ✗  $BCC$  ✗  $CDC$  ✓



### AlternateSuccession(A, B)

LTL Formalization:  $\Box (\mathbf{A} \rightarrow \mathbf{O}(\neg \mathbf{A} \mathbf{U} \mathbf{B})) \wedge (\neg \mathbf{B} \mathbf{W} \mathbf{A}) \wedge \Box (\mathbf{B} \rightarrow (\neg \mathbf{B} \mathbf{W} \mathbf{A}))$

$A$  and  $B$  occur in the process instance if and only if the latter follows the former, and they alternate each other in the trace.

$BCAAC$  ✗  $CAACB$  ✗  $CACB$  ✓  $CABCA$  ✗  
 $BCC$  ✗  $CACBAB$  ✓



### ChainSuccession(A, B)

LTL Formalization:  $\Box (\mathbf{A} \rightarrow \mathbf{O} \mathbf{B}) \wedge \Box (\mathbf{O} \mathbf{A} \rightarrow \mathbf{B})$

$A$  and  $B$  occur in the process instance if and only if the latter immediately follows the former

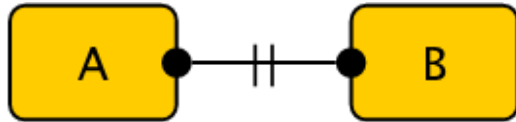
$BCAAC$  ✗  $BCAABC$  ✗  $CABABC$  ✓





# DECLARE constraint templates

## Relation templates



### NotCoExistence(A,B)

LTL Formalization:  $(\diamond A \rightarrow \neg \diamond B) \wedge (\diamond B \rightarrow \neg \diamond A)$

A and B never occur together in the process instance

CAC ✓ CAACB ✗ BCAC ✗ BCC ✓ CDC ✓

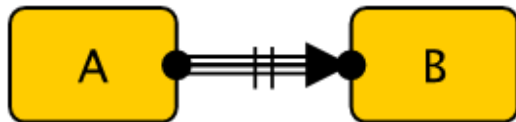


### NotSuccession(A,B)

LTL Formalization:  $\square (A \rightarrow \neg \diamond B)$

A can never occur before B in the process instance

BCAAC ✓ CAACB ✗ CAC ✓ BCC ✓



### NotChainSuccession(A,B)

LTL Formalization:  $\square (A \rightarrow \neg OB)$

A and B occur in the process instance if and only if the latter does not immediately follows the former

BCAAC ✓ BCAABC ✗ CBACBA ✓

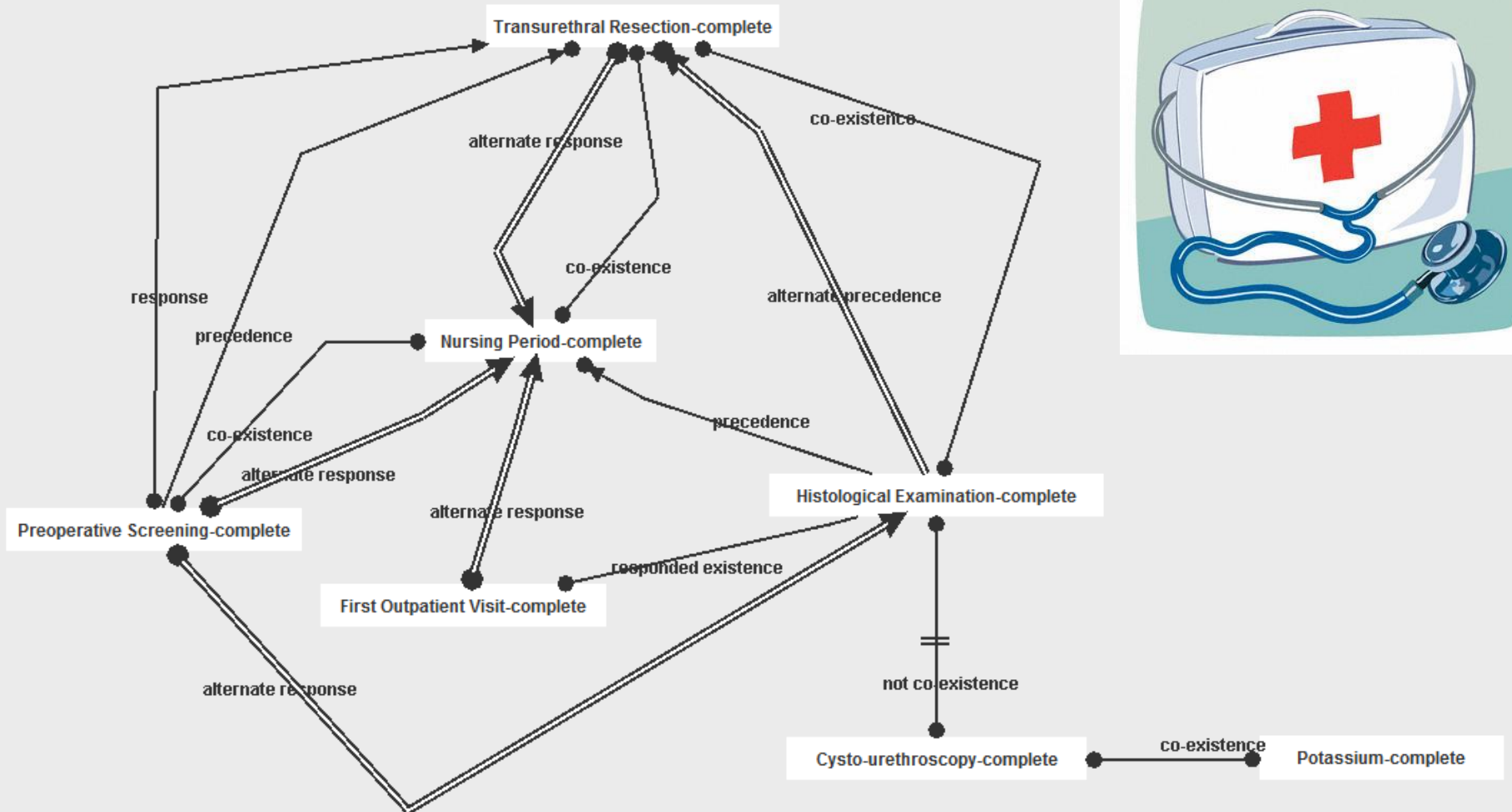


# The DECLARE System

- The **DECLARE System** consists of the Designer, the Framework, and the Worklist.
  - The **DECLARE Designer** consists of a graphical editor component for creating and verifying DECLARE models.
  - The **DECLARE Framework** works as the backend server for executing DECLARE processes.
  - The **DECLARE Worklist** is the user client connecting to the Framework.
- The DECLARE System can be downloaded from:  
<http://www.win.tue.nl/declare/download/>

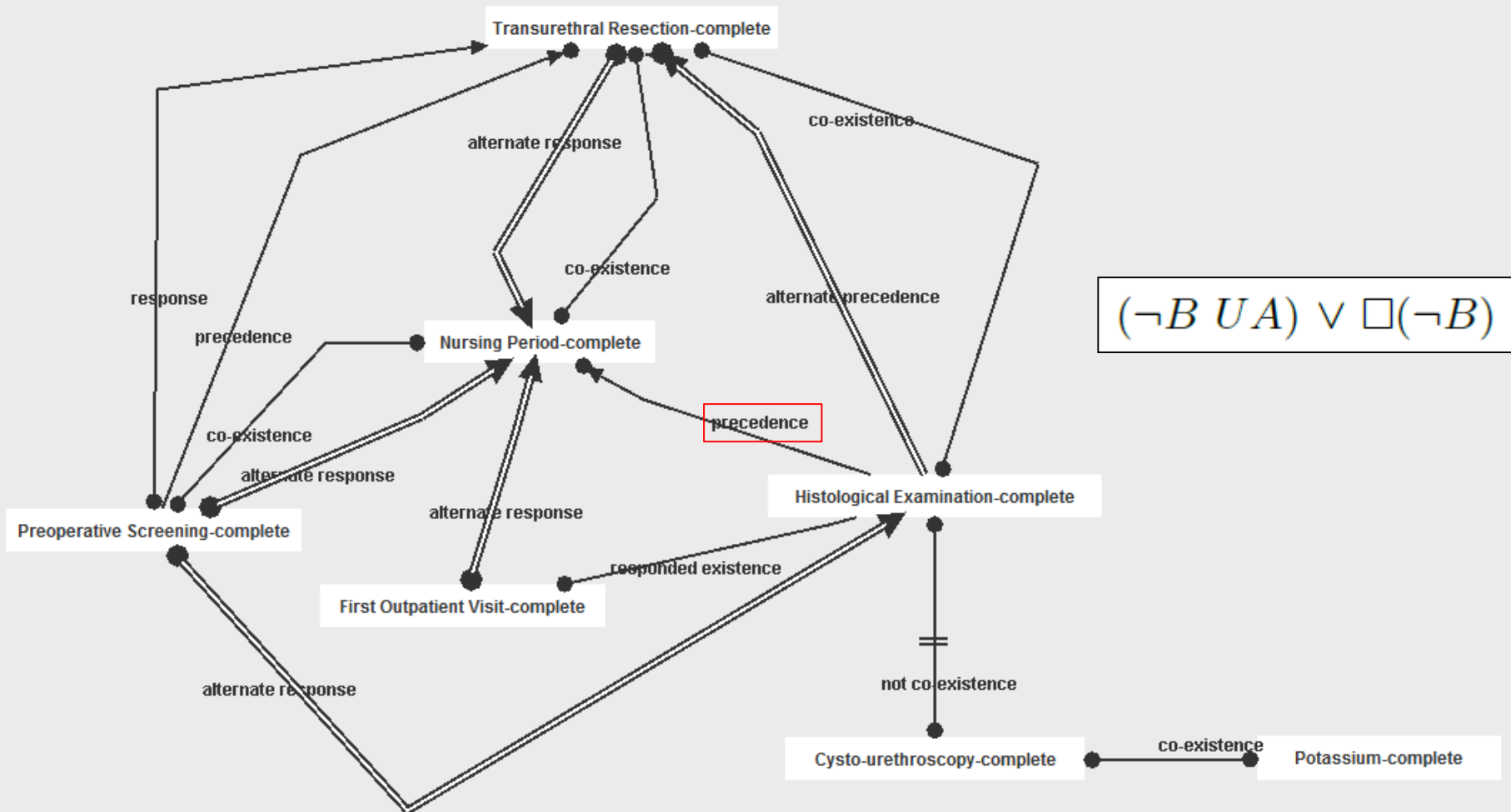


# An Example of DECLARE Model



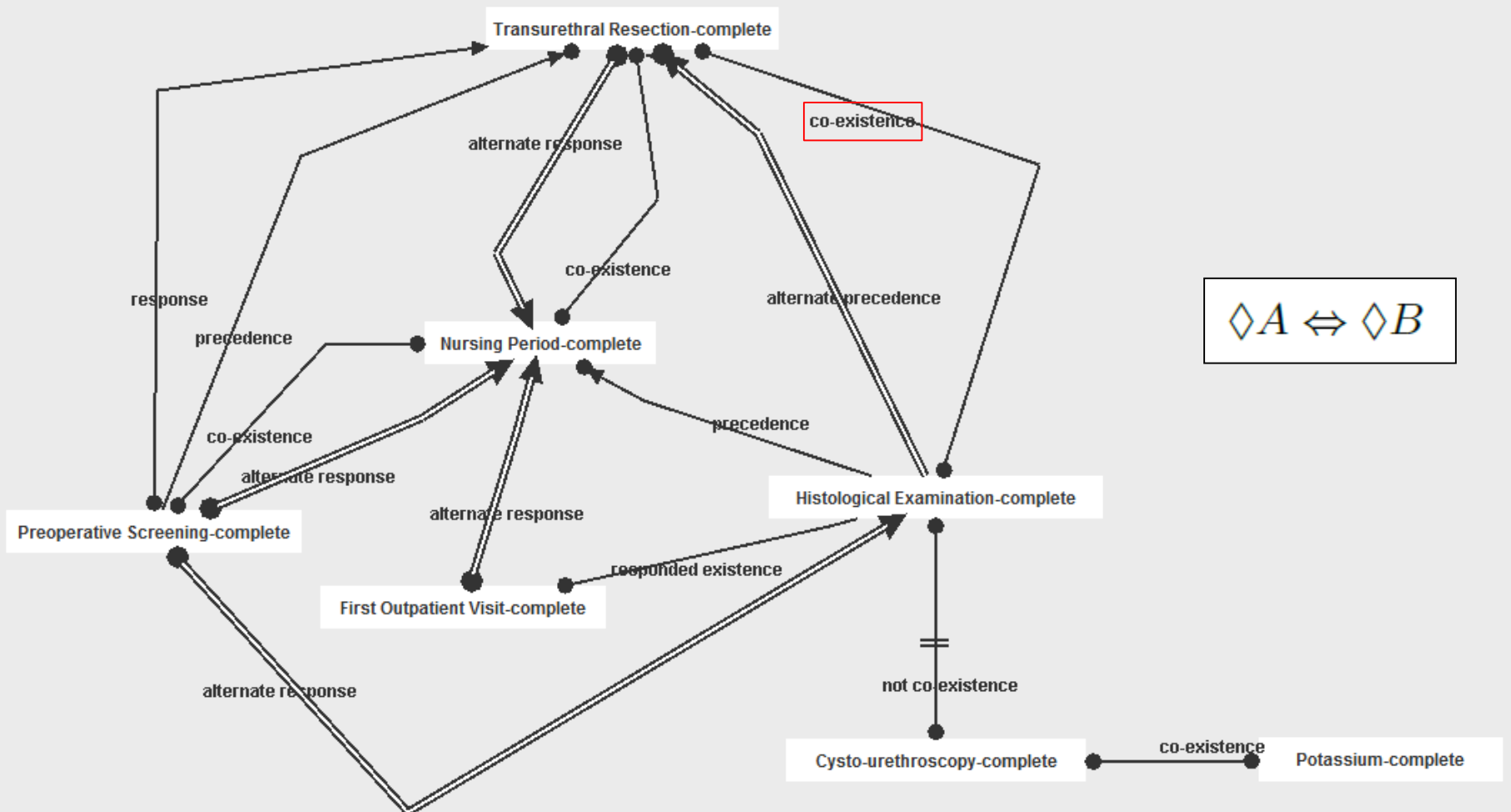


# An Example of DECLARE Model



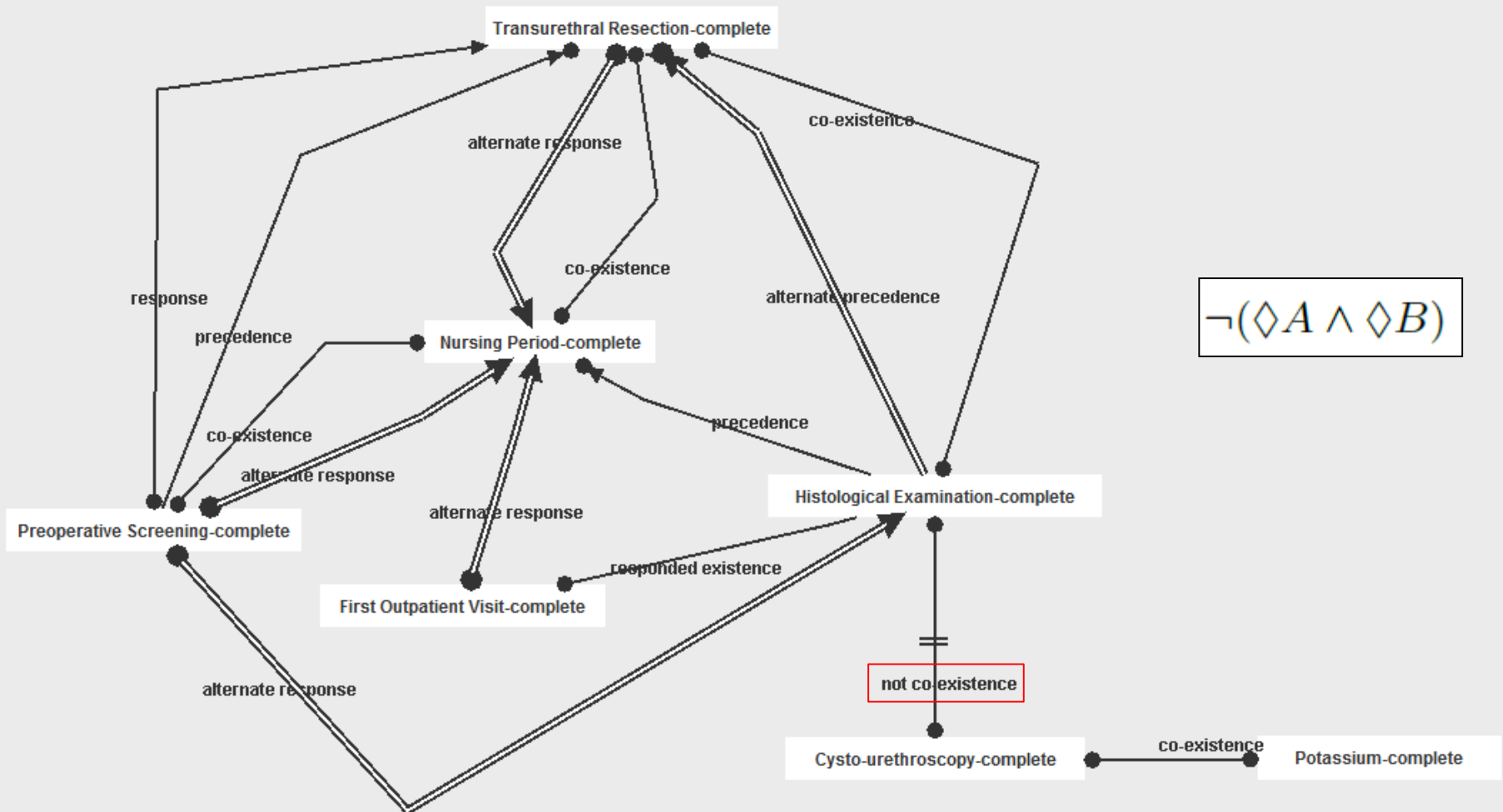


# An Example of DECLARE Model





# An Example of DECLARE Model





# Declarative Process Mining in ProM

- Declare Maps Miner
- Declare Analyzer
- Declare Replayer
- Declare Diagnoser
  - and many others....for a complete list, check:

Fabrizio Maria Maggi

***Declarative Process Mining with the Declare Component of ProM***

12th International Conference on Business Process Management,  
BPM 2014



# Declare Maps Miner /1

- The **Declare Maps Miner** allows to generate from scratch a set of DECLARE constraints representing the actual behavior of a process as recorded in an event log.
- The user selects from a list of DECLARE templates the ones to be used for the discovery task.
  - The mined model will contain **only constraints** that are instantiations of the selected templates.





# Declare Maps Miner /2

## Declare Maps Miner Plugin

### Choose the Templates to Mine

Template
coexistence
absence
responded existence
existence
chain precedence
precedence
not coexistence
exactly2
not chain succession
exactly1
alternate precedence
<b>chain succession</b>
succession
absence2
...

**Select All**  
**Deselect All**  
**Add**  
**Remove**

**Selected Templates**  
chain succession  
coexistence  
existence  
precedence

**Description**

**chain succession**

A and B can happen only next to each other.

`[](( "A" = X( "B" )))`



# Declare Maps Miner /3

The user can **ignore constraints between event types** of the same activity (i.e., involving different parts of the activities' lifecycle such as *start* and *complete*).

The user can **clusterize different activities in different groups** and specify if only *intra-group* or *inter-group* constraints should be considered. For example, in a hospital log, an analyst would be interested in constraints between activities involved in surgery and therapy.

## Declare Maps Miner Plugin

### Apriori and Activation/Satisfaction Configuration

- All Activities (considering Event Types)
- All Activities (ignoring Event Types)
- Diversity (Ignore associations between event types of same activity)
- Concept Based associations

#### Concept Based Item Set Configuration

Load Concept Ontology  ..

- Intra-Group Concept Associations
- Inter Group Concept Associations

#### Configure support/alpha

Choose support 100 if your log contains no noise

Min. Support  64

Choose alpha as 0 if you want to discover only those constraints that are always activated in the log (non-trivially true)

Alpha  28



# Declare Maps Miner /4

## Declare Maps Miner Plugin

### Apriori and Activation/Satisfaction Configuration

- All Activities (considering Event Types)
- All Activities (ignoring Event Types)
- Diversity (Ignore associations between event types of same activity)
- Concept Based associations

#### Concept Based Item Set Configuration

Load Concept Ontology

- Intra-Group Concept Associations
- Inter Group Concept Associations

#### Configure support/alpha

Choose support 100 if your log contains no noise

Min. Support  64

Choose alpha as 0 if you want to discover only those constraints that are always activated in the log (non-trivially true)

Alpha  28

The user can also **specify thresholds** for parameters *minimum support* and *alpha*.

**Minimum support** allows to select the percentage of traces in which a constraint must be satisfied to be discovered (and to filter out noisy traces).

**Alpha** can be used to ignore constraints that are trivially true in the discovery task.  
*For example, constraint  $response(A,B)$  is trivially true in process instances in which A does not occur at all.*



The discovery results are presented to the user as **interactive maps**. Activities are colored based on their frequency (from **white** indicating low frequency to **yellow** indicating high frequency).

Mined Model

The screenshot displays the 'Mined Model' interface. On the left, a process flow diagram shows activities such as 'incoming claim', 'B check if sufficient information is available', 'S check if sufficient information is available', 'determine likelihood of claim', 'close claim', 'assess claim', 'initiate payment', and 'advise claimant on reimbursement'. Activities are connected by 'response' edges. Some activities are highlighted in yellow, indicating higher frequency. On the right, a panel shows a list of activities with their support, CPIR, and Confidence values. Below this is a 'Filtering' section with checkboxes for 'response', 'existence', and 'absence'. At the bottom right, there are sliders for 'Support', 'Confidence', 'CPIR', and 'IF', and a 'Regenerate Model' button. At the bottom left, statistics are shown: 'Number of activities in this map: 11' and 'Number of constraints in this map: 20'.

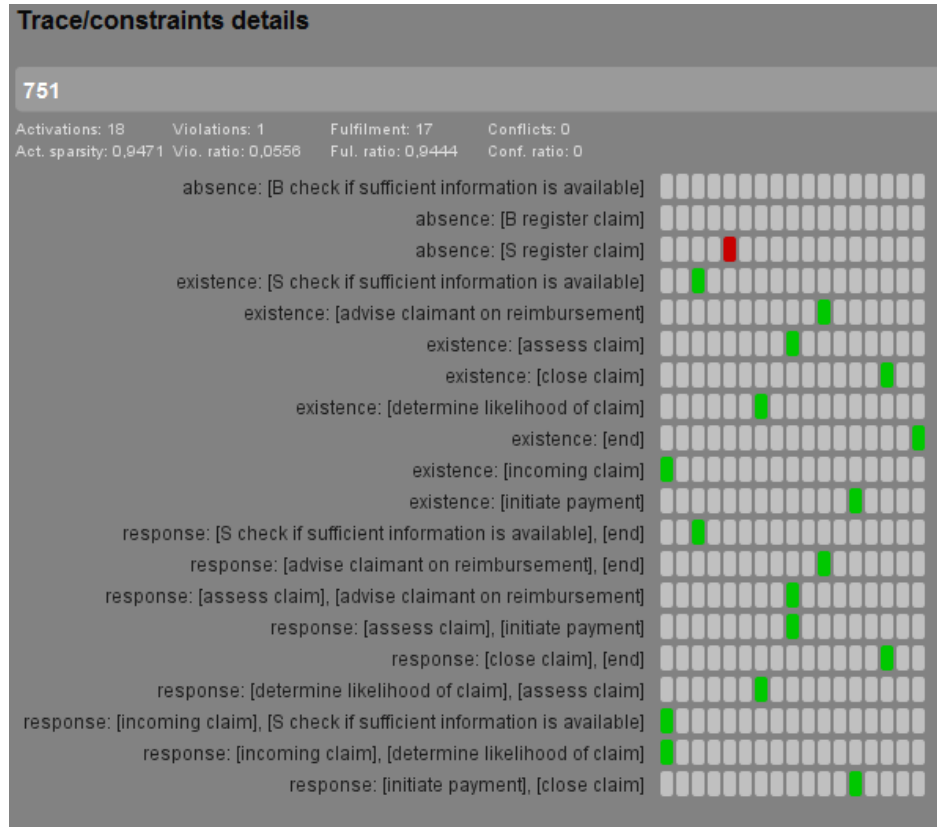
Number of activities in this map: 11  
Number of constraints in this map: 20

The user can **prune out**, in the discovered maps, the constraints that are less interesting, redundant or deriving from noise in the log.



# Declare Analyzer

- Through the **Declare Analyzer** the user can *pinpoint* where the process execution *deviates* from the reference DECLARE model.
- The **degree of conformance** of the process behavior can be quantified through several metrics, e.g., fulfillment ratio and violation ratio.





# Declare Replayer

- The **Declare Replayer** generates a set of ***alignments*** between the log and the reference DECLARE model, i.e., information about what must be changed in the log traces to make them perfectly compliant with the model.

The screenshot displays the 'Visualization data' interface of the Declare Replayer. The main panel shows 'Log-Model Alignments. Average Fitness 0.95'. It lists three traces:

- Trace 8**: Detail, 1.0 fitness, 5 moves, represented by five green chevrons.
- Trace 1**: Detail, 0.8 fitness, 3 moves, represented by three green chevrons.
- Trace 2**: Detail, 0.8 fitness, 3 moves, represented by three green chevrons.

A legend on the right side defines the alignment types:

- Green chevron: Perfect Alignment Step (Move log and model)
- Purple chevron: Missing Event (Move model only)
- Yellow chevron: Wrong Event (Move log only)

Below the main panel, a section titled 'Details of the Alignment for Trace 3' shows three green bars representing alignment steps:

- 1: A-complete**: Contributes to Solve 6 Violations of Constraints
- 2: B-complete**: Contributes to Solve 8 Violations of Constraints
- 3: C-complete**: Contributes to Solve 8 Violations of Constraints



# Declare Diagnoser

- The **Declare Diagnoser** projects the results obtained through the Declare Replayer onto the reference model.
- This projection produces a map in which the **critical activities/constraints** of the reference model are **highlighted**.
- Activities are colored from **red** to **green** according to the number of moves they are involved in, i.e., according to how many times they were in the wrong place in the log or missing when required.
- Constraints are colored from **red** to **green** based on the number of moves that are needed to make the log compliant to them.

