

- Expression of properties
 - on BPMN models
 - **Valery MONTHE**
 - ENSTA Bretagne

Plan

1. Context
2. Issue
3. Objective et approach
4. Environment and execution scenario
5. Implementation
6. Summary

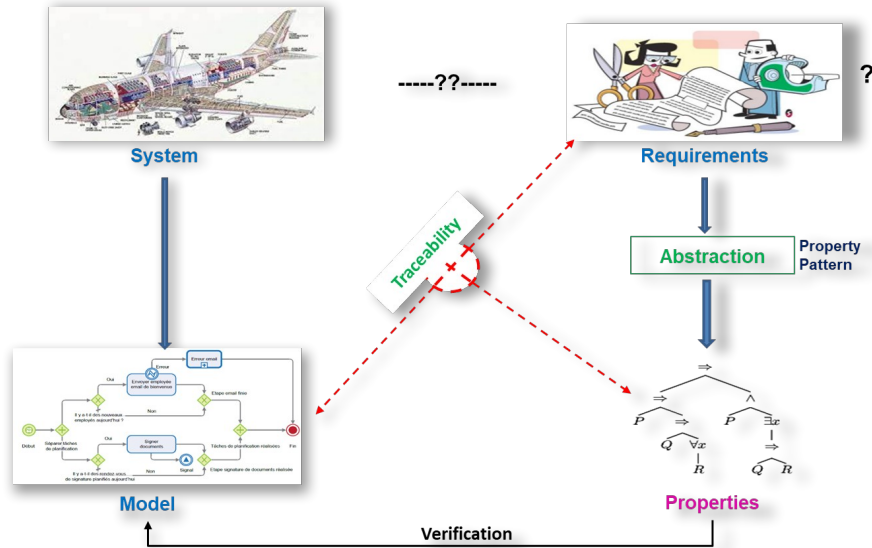
Issue

- **An illustrative example**
 - **Requirement**
 - The production of component A leads to the production of component B
 - Each time component A is produced, component B will be
 - **Property in temporal logic**
 - P : component A is produced (**atomic proposition**)
 - S : component B is produced (**atomic proposition**)
 - $[] (P \rightarrow \langle \rangle S)$ *LTL*
 - $AG (P \rightarrow AF (S))$ *CTL*
- **Another example**
 - $[] (Q \rightarrow ((P \rightarrow (!R \cup S)) \cup R) \mid [] (P \rightarrow (!R \cup S)))$
 - $AG (Q \rightarrow !E [!R \cup (P \ \& \ !R \ \& \ (E [!S \cup R] \mid EG (!S \ \& \ !R)))])$

- ❑ **Formal language and temporal logic**
 - A formalism far from business experts
 - Several temporal logic : **LTL, CTL, TCTL, etc.**
 - ❑ **Monitoring of requirements : traceability**
 - Link between program objectives, requirements and properties

Issue and approach

- **Two questions to answer :**
 - How to define verifiable properties in temporal logic ?
 - How to link requirement, property and model?



- **Approach**

- **A common representation**
 - Raising the Level of Abstraction : From Requirements to Language Independent Formulation
- **Heterogeneous modeling: requirements, pattern, properties and PDP model**

Dwyer's pattern

- **An illustrative example**
 - **Requirement**
 - The production of component A leads to the production of component B
 - Each time component A is produced, component B will be

 - **Property in temporal logic**
 - P : component A is produced (atomic proposition)
 - S : component B is produced (atomic proposition)

 - $[] (P \rightarrow \langle \rangle S)$ LTL => **Globaly S Responds to P**
 - $AG (P \rightarrow AF (S))$ CTL

- **Another example**
 - $[] (Q \rightarrow ((P \rightarrow (!R U S)) U R) \mid [] (P \rightarrow (!R U S)))$ => **After Q until R (S Responds to P)**

Dwyer's Pattern

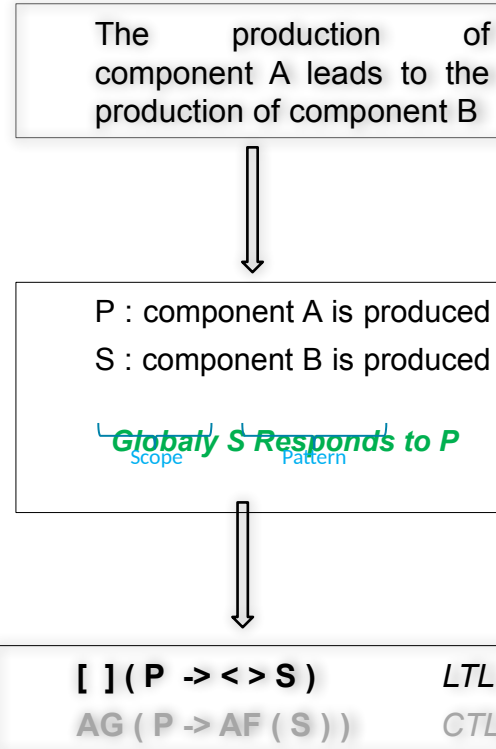
- Requirement



- Abstraction
Dwyer's Pattern



- LTL Property
(temporal logic)



Modeling environment

Heterogeneous modeling platform : Oneway Properties Editor

An environment that allows you to :

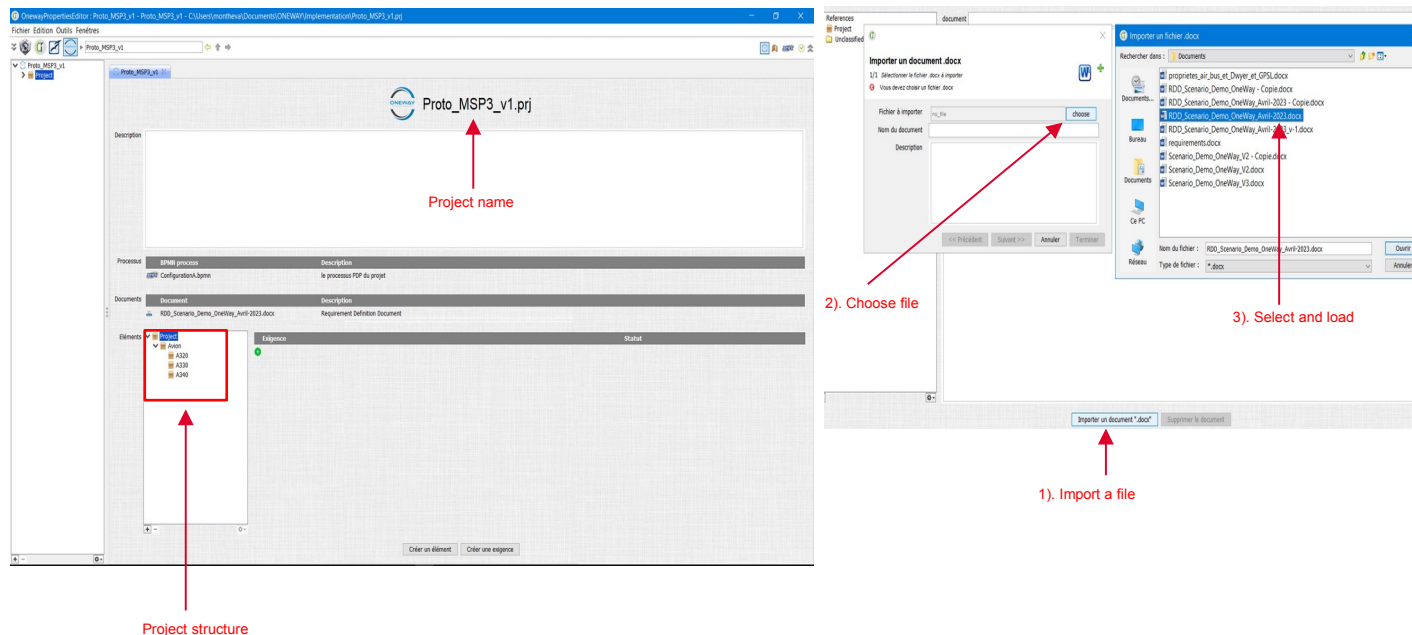
1. Load and read the requirements definition document (Word, Excel, pdf, etc.)
2. Create requirements and link them to document content
3. Load the models of the processes to check
4. Define the properties verifying each requirement, in a language close to the natural
5. Generate the temporal logic formula for each property
6. Produce a file for the model checker



Usage scenario

1. Create a project, structure it and upload the requirements definition document

- A project can be structured in subsystems (elements)
- Requirements definition document (program objectives) written by business experts



Usage scenario

2. Create requirements and link to the requirements definition document

- Instantiation of the requirements contained in the document
- Instantiating relationships between a requirement and its specification contained in the document

Requirements Definition Document

Open document name

Possible actions between document and requirements

Set up the requirement

Create a requirement

Semantics of a requirement linked to element Req_1

Link between Req_1 and the text

Document content

Documents: Document, Description: Requirement Definition Document

Req_1: Propriétés de OneWay en langage naturel
 ==> Semantique et spécification de la propriété

Req_2: Un programme qui commence doit toujours terminer
 ==> S'il y a un Start, on veut toujours un Finish
 Un Start est toujours suivi d'un Finish

Req_3: Un processus terminé doit avoir son graphe propre : pas d'activités restantes, pas de signal à émettre ou attendu, etc.
 ==> Si le processus est terminé, cela signifie que tout est fini. Plus de tokens.
 Si Finish alors graphe is Cleared

Req_4: L'élément de fin d'un processus doit être la dernière tâche à exécuter
 ==> Si le processus est terminé, cela signifie que tout est fini. Plus de tokens.
 Si Finish alors graphe is Cleared

Req_5: Quand on atteint la fin du processus, tous les signaux doivent avoir été envoyés et reçus
 ==> Si le processus est terminé, cela signifie que tout est fini. Plus de tokens.
 Si Finish alors graphe is Cleared

Req_6: Quand on atteint la fin du processus, toutes les activités doivent avoir été exécutées.
 ==> Si le processus est terminé, cela signifie que tout est fini. Plus de tokens.
 Si Finish alors graphe is Cleared

Req_7: Chaque programme a un time to market qui ne doit pas être dépassé.
 ==> Si on est à finish alors le temps écoulé <TIM

Req_8: S'il y a le livrable A alors un jour il y aura le livrable C
 ==> L'événement A est toujours suivi de l'événement C

Usage scenario

3. Defining predicates: atomic propositions

- Load an instance of the PDP (BPMN model)
- Define predicates on the execution model

1) Import file

2) Choose file

3) Select the file and open

4) Click to create atomic proposition

5) Proposition to create

6) Select execution model

7) Choose the predicate to evaluate

8) Expression to check

9) Defined atomic propositions

Proposition atomique	Expression
start	processExecution.isStarted()
finish	processExecution.isEnded()
canEnd	((!(start) finish)
clear	processExecution.isCleared()

Usage scenario

4. Define properties to verify

- Combine atomic propositions into an abstract formula (Dwyer's pattern)

1) Create the property

Nouvelle proposition atomique Nouvelle propriété

Créer une propriété 2) Step 1/3

Propriété: p2

requirement: Req_2

Description: Requirements available

Requirements available

3) Step 2/3

Créer une propriété

2/3 Choisir un scope

Scope de la propriété

- Globally
- Before
- After
- Between
- AfterUntil

Choose scope

4) Step 3/3

Créer une propriété

3/3 Choisir un pattern

Pattern

- Absence
- Universality
- Existence
- BoundedExistence
- Precedence
- Response
- ChainPrecedence
- ChainResponse

Choose pattern

Expression: finish

Expression 2: start

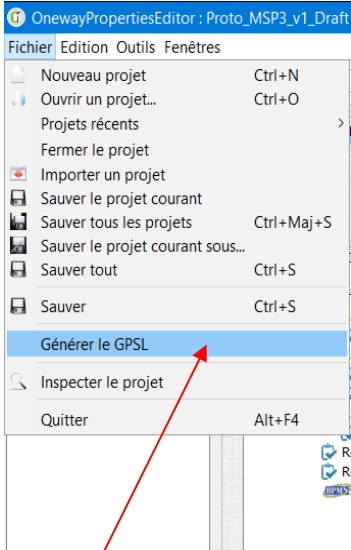
5) Defined properties

Propriété	Expression
p1	globally universality canEnd
p2	globally finish response start
p3	globally universality ((!(finish)) clear)

Expression of the property obtained

Usage scenario

5. Generation of the GPSL properties file: properties verified by OBP



Generate the GPSL

GPSL File Contents

```

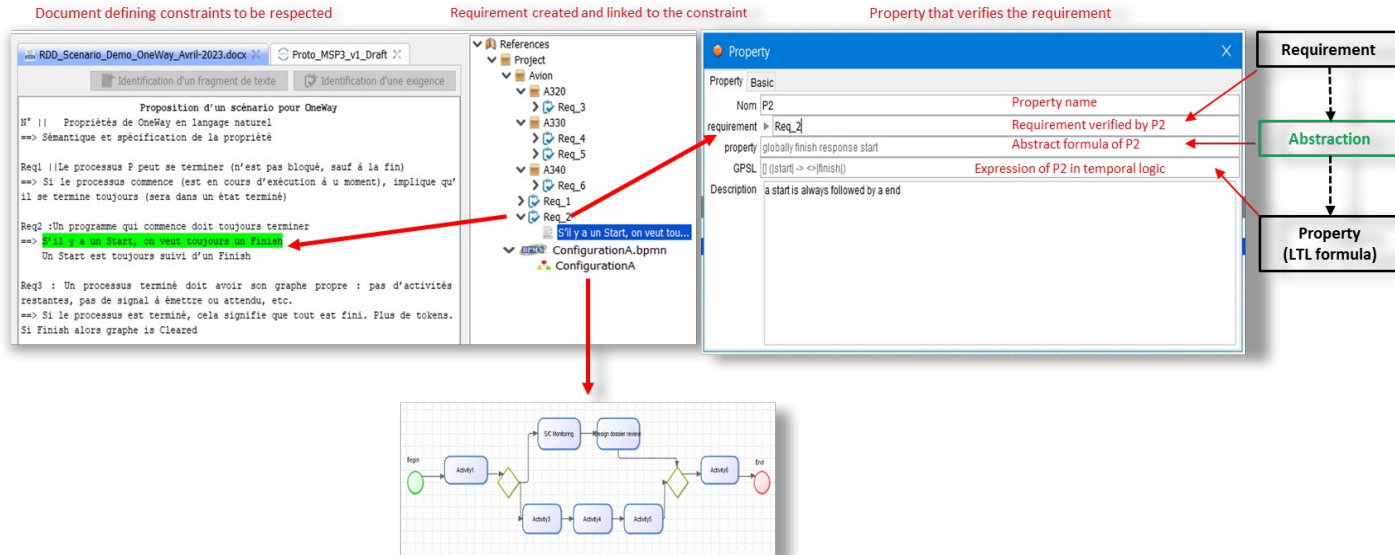
1 //-----
2 //-----GPSL file-----
3 //-----
4 // this file contains all the properties defined for your projet
5
28 //-----Atomic proposition N°:3-----
29 // Name : canEnd
30 // Description : the process can end. that is start=>finish
31
32 let canEnd=((!(start)) | finish)
33 //-----Atomic proposition N°:4-----
34 // Name : clear
35 // Description : return True if the process is clear. that is no more tokens left in process execution
36
37 let clear=processExecution.isCleared()
38
39 //-----List of properties-----
40 //---Number of properties : 3
41 //-----
42 //-----Property N°:1-----
43 // Name : p1
44 // Requirement :
45 // Req_1 : Le processus P peut se terminer (n'est pas bloqué, sauf à la fin)
46 // GPSL :
47
48 p1 = [] (|canEnd|)
49 //-----Property N°:2-----
50 // Name : P2
51 // Requirement :
52 // Req_2 : Un programme qui commence doit toujours terminer
53 // GPSL :
54
55 P2 = [] (|start| -> <>|finish|)

```

Usage scenario

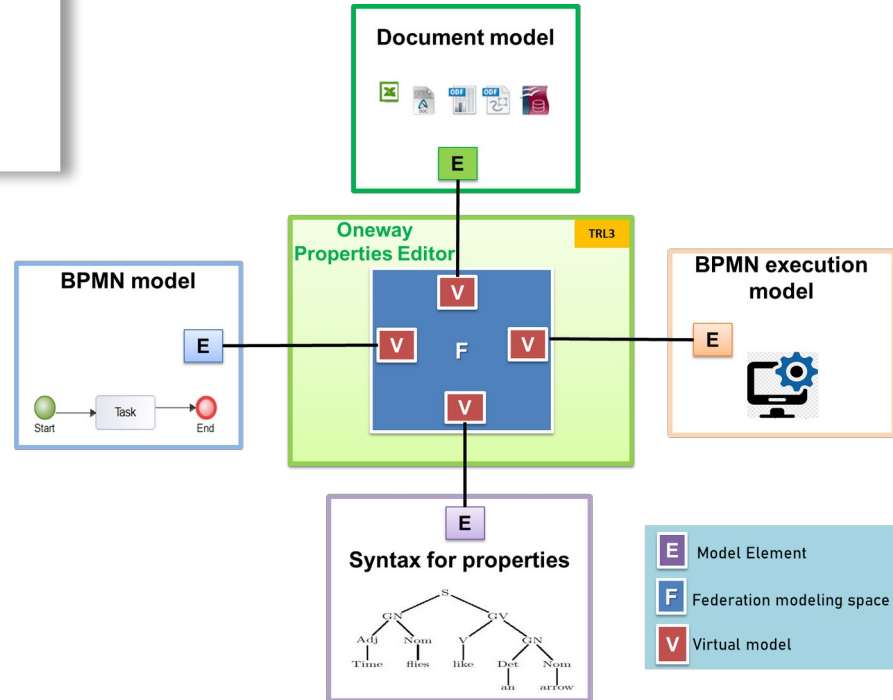
6. summarize : end-to-end traceability

- Requirement : « **a program that starts must always finish** »
- Propositions to be evaluated: « **Start** : the program has started» et « **Finish** : the program is finished»
- Abstract definition of property : « **Globaly Finish Response To Start** »
- Generated temporal logical property « **[] (|Start| -> <> |Finish|)** »



Conceptualization : Federation

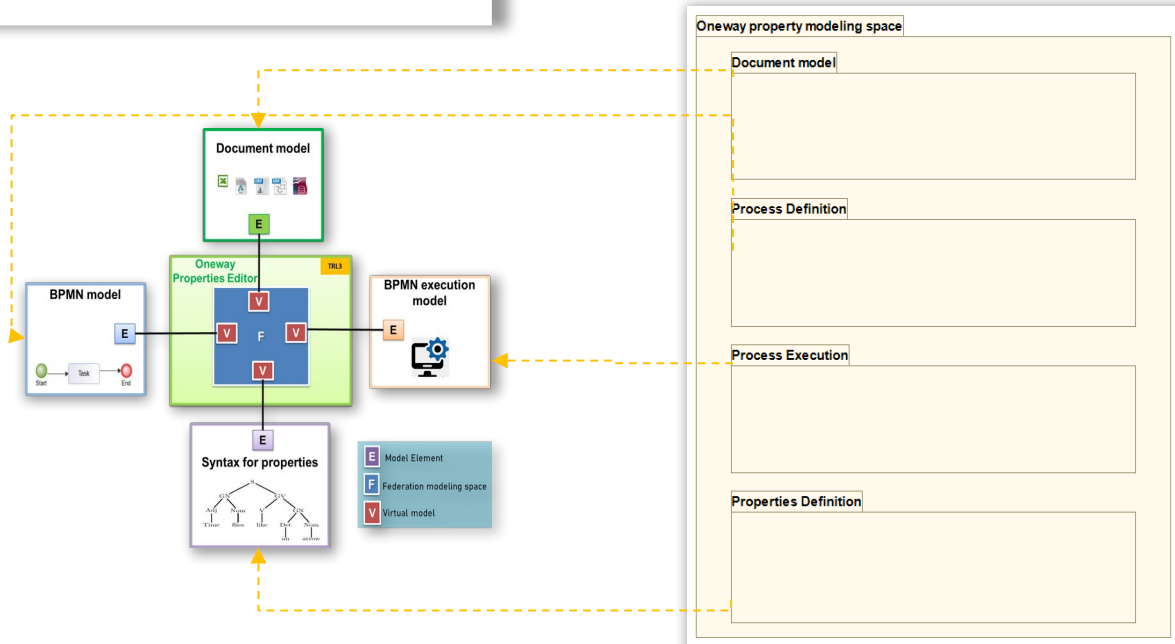
- Federate several modelling spaces:
 - ✓ BPMN model
 - ✓ BPMN execution model
 - ✓ Predicate language
 - ✓ Syntax for properties definition
 - ✓ Document model



Conceptualization : Federation

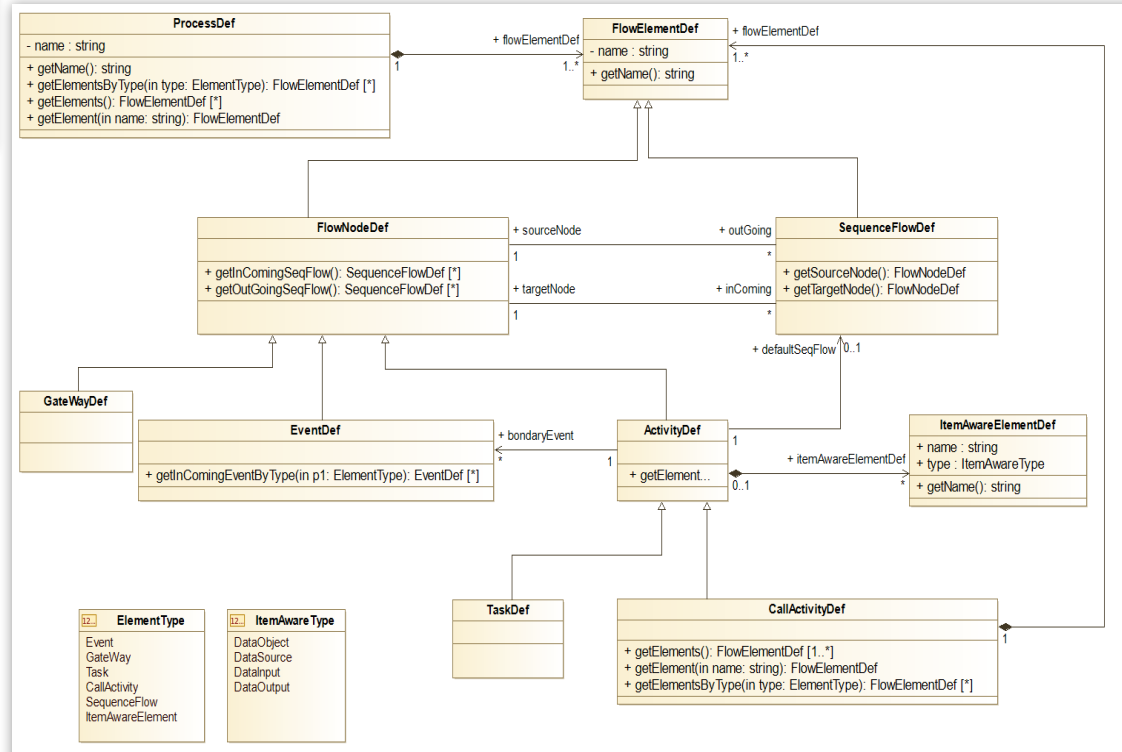
▪ **A unique space to manipulate heterogeneous models**

- ✓ BPMN model
- ✓ BPMN execution model
- ✓ Predicate language
- ✓ Syntax for properties definition
- ✓ Document model



Conceptualization : Process model

- A process definition model :
 - ✓ An extract from the BPMN meta-model
 - ✓ Key concepts for manipulating process elements
 - ✓ Operations to query a process



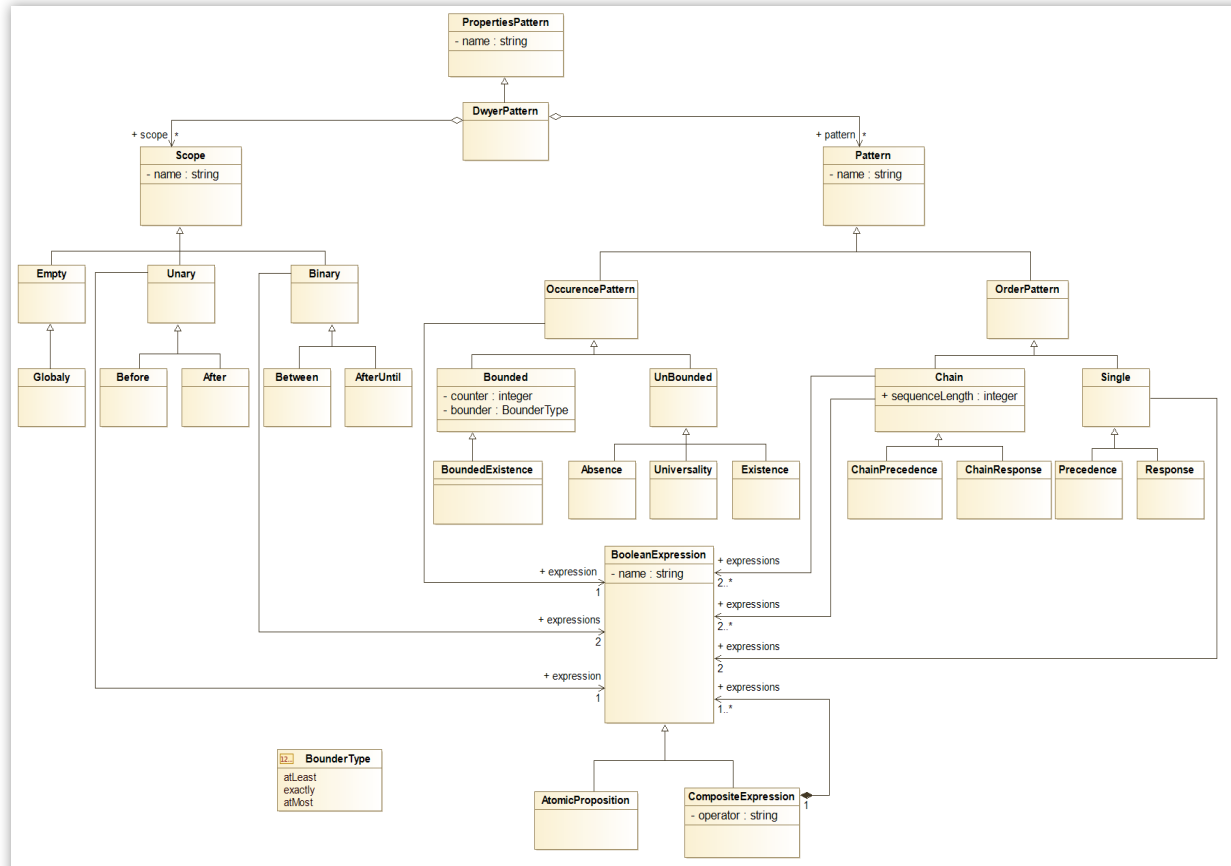
Conceptualization : Process execution

- A process execution model :
 - ✓ Token management for process execution
 - ✓ Methods to query/browse a process execution
 - ✓ Predicates defined using these methods



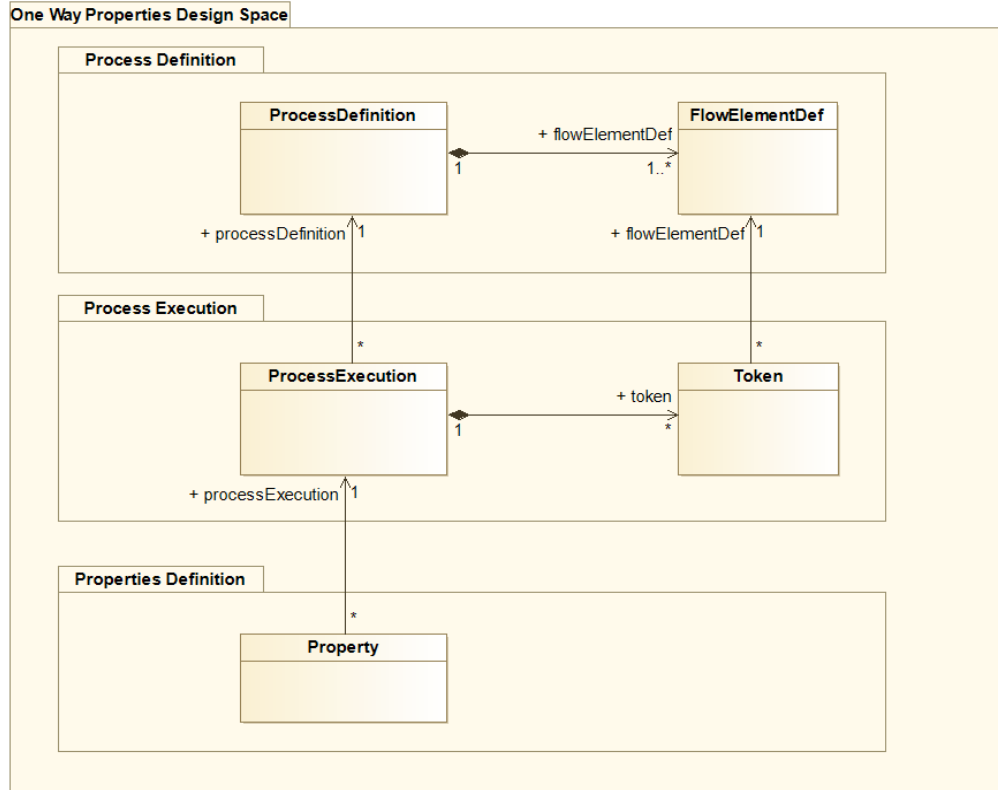
Conceptualization : Property definition

- A properties definition model :
 - ✓ Use of Dwyer patterns
 - ✓ Syntax for defining properties
 - ✓ Expression of properties using predicates and Dwyer patterns



Conceptualization : Model interaction

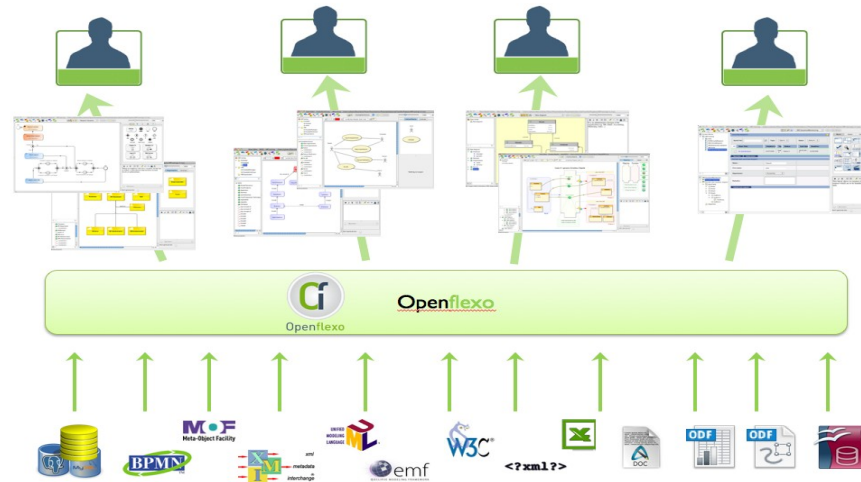
- Properties are set on a process execution
- Execution model manages a set of tokens related to process elements



Toolchain : Openflexo

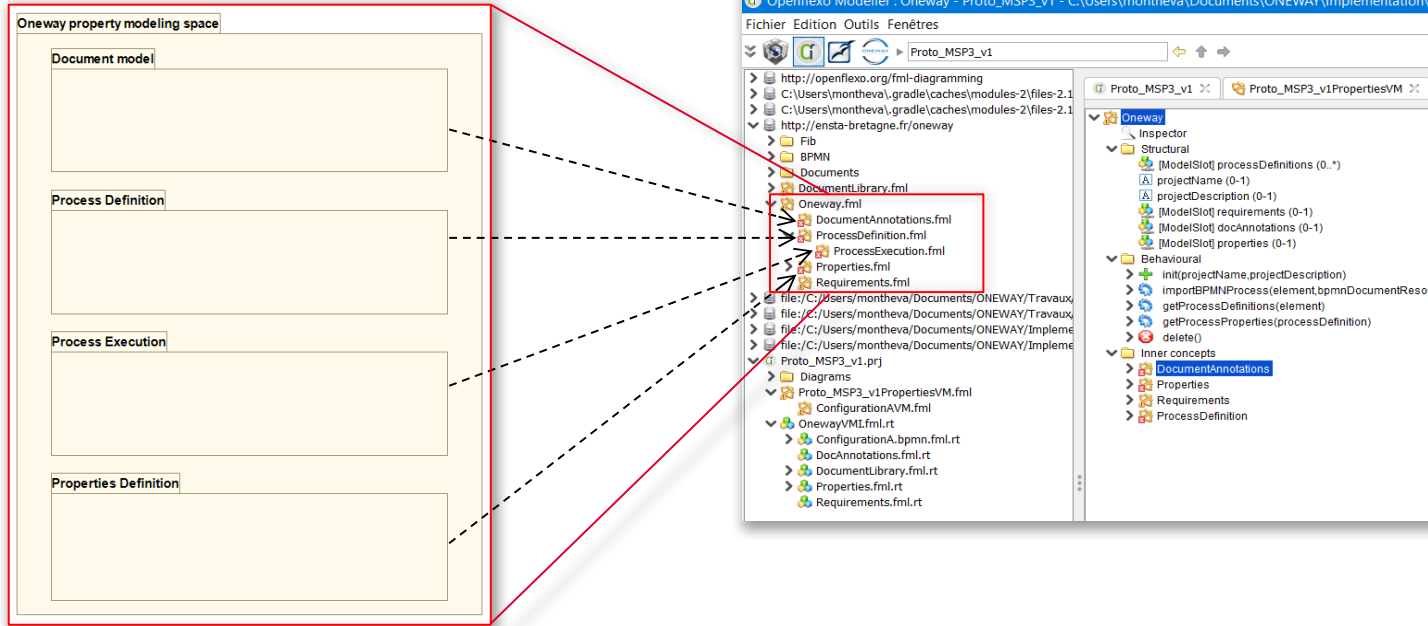
A model federation framework

- Connect heterogeneous models
- Build new concepts from existing concepts
- Design a custom representation for these concepts
- Provide tools to manipulate these elements



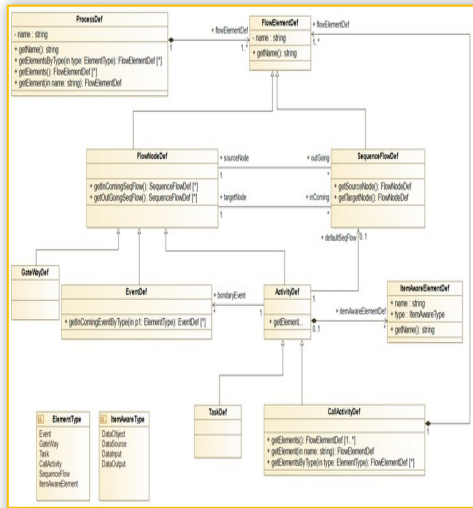
Implementation : Modelling space

- A modelling space with several subspaces
- Connecting heterogeneous models



Implementation : Process definition

- Implementation of an excerpt of the BPMN metamodel



Openflexo Modeller : ProcessDefinition - Proto_MSP3_v1 - C:\Users\montheva\Documents\ONEWAY\Implementation

Fichier Edition Outils Fenêtres

Proto_MSP3_v1

- http://openflexo.org/fml-diagramming
- C:\Users\montheva\gradle\caches\modules-2\files-2.1
- C:\Users\montheva\gradle\caches\modules-2\files-2.1
- http://ensta-bretagne.fr/oneway
 - Fib
 - BPMN
 - Documents
 - DocumentLibrary.fml
 - Oneway.fml
 - DocumentAnnotations.fml
 - ProcessDefinition.fml
 - ProcessExecution.fml
 - Properties.fml
 - Requirements.fml
- file:/C:/Users/montheva/Documents/ONEWAY/Travaux
- file:/C:/Users/montheva/Documents/ONEWAY/Travaux
- file:/C:/Users/montheva/Documents/ONEWAY/Impleme
- file:/C:/Users/montheva/Documents/ONEWAY/Impleme
- Proto_MSP3_v1.prj
 - Diagrams
 - Proto_MSP3_v1PropertiesVM.fml
 - ConfigurationAVM.fml
 - OnewayVMI.fml.rt
 - ConfigurationA.bpmn.fml.rt
 - DocAnnotations.fml.rt
 - DocumentLibrary.fml.rt
 - Properties.fml.rt
 - Requirements.fml.rt

Inspector

- Structural
 - processName (1)
 - [ModelSlot] bpmnEditor (0-1)
 - element (0-1)
 - description (0-1)
- diagrams
- Behavioural
 - getName()
 - getElements()
 - getElement(name)
 - getElementsByType(elementType)
 - init(element bpmnResource, bpmnName, bpmnDescription)
 - delete()
 - getEvent(name)
- Inner concepts
 - ProcessExecution
 - FlowNodeDefinition
 - FlowElementDefinition
 - ElementType
 - SequenceFlowDefinition
 - ItemAwareType
 - ItemAwareElementDefinition
 - ElementStatusType

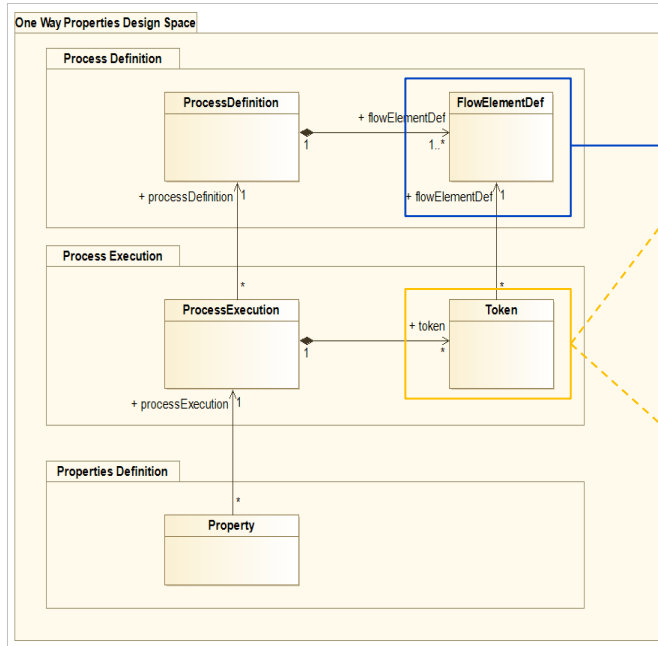
Implementation : Process execution

- Implementation of predicates
- These are evaluable logical expressions
- Uses tokens to query a model run

The screenshot shows the Openflexo Modeller interface. On the left, a class diagram shows the `ProcessExecution` class with attributes like `name`, `status`, and `tokens`, and methods like `getTokens()` and `isRunning()`. The center pane shows a file explorer with the project structure, highlighting `ProcessExecution.fml`. The right pane shows the `Inspector` for the `ProcessExecution` object, listing various methods. A red box highlights the `isRunning()` method's implementation: `return ((this.getTokensByType(null).size() = 0) & (this.tokens.size > 0))`. A callout box on the right, labeled "Evaluable logical expression", points to this code. Below the callout, a small table shows the expression type and the full logical expression: `((this.getTokensByType(null).size() = 0) & (this.tokens.size > 0))`.

Implementation : Process execution

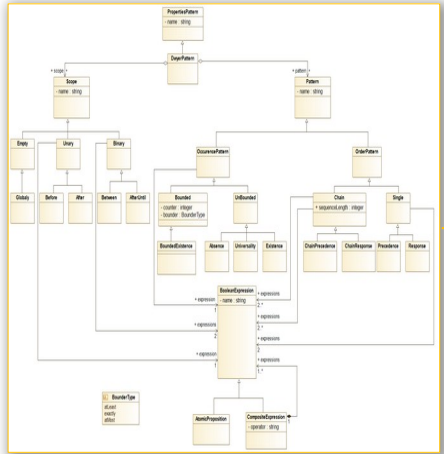
- A token refers to an element of the process definition



The screenshot shows a software interface with two panes. The left pane is a tree view under 'Token' with sub-items: 'Inspector', 'Structural', 'element (1)', 'Behavioural', 'delete()', 'getElement()', and 'Inner concepts'. A blue arrow points from 'element (1)' to the right pane. The right pane shows the 'Basic' tab of a 'FlexoConcept' instance. It lists properties: 'Type' (FlexoConceptInstanceRole), 'Nom' (element), 'Visibilité' (Défaut), 'technology' (FML@runtime technology adapter), 'virtual_model_instance' (processDefinition), 'FlexoConcept' (FlowElementDefinition), 'Cardinalité' (1), 'Conteneur', 'Valeur par défaut', and 'Requis' (checkbox). A red box highlights 'processDefinition' and 'FlowElementDefinition'. At the bottom, it shows '0 errors, 0 warnings' and an 'errors/warnings' section.

Implementation : Property definition

- An abstraction based on Dwyer's patterns
- Designing a Dwyer's pattern metamodel



Openflexo Modeller : Properties - Proto_MSP3_v1 - C:\Users\montheva\

Fichier Edition Outils Fenêtres

Proto_MSP3_v1

- http://openflexo.org/fml-diagramming
- C:\Users\montheva\gradle\caches\modules-2\files-2.1\org.openflexo\http
- C:\Users\montheva\gradle\caches\modules-2\files-2.1\org.openflexo\moc
- http://ensta-bretagne.fr/oneway
 - Fib
 - BPMN
 - Documents
 - DocumentLibrary.fml
 - Oneway.fml
 - DocumentAnnotations.fml
 - ProcessDefinition.fml
 - ProcessExecution.fml
 - Properties.fml
 - ProcessProperties.fml
 - Requirements.fml
- file:/C:/Users/montheva/Documents/ONEWAY/Travaux/BPMN/Ressources/
- file:/C:/Users/montheva/Documents/ONEWAY/Travaux/
- file:/C:/Users/montheva/Documents/ONEWAY/Implementation/Documents
- file:/C:/Users/montheva/Documents/ONEWAY/Implementation/BPMN/
- Proto_MSP3_v1.prj
- Diagrams

ProcessProperties.fml

- AtomicProposition
- Property
- Scope
 - Globally
 - UnaryScope
 - Before
 - After
 - BinaryScope
 - Between
 - AfterUntil
- Pattern
 - OccurrencePattern
 - UnboundedOccurrencePattern
 - Absence
 - Universality
 - Existence
 - BoundedExistence
 - OrderPattern
 - SingleOrderPattern
 - Precedence
 - Response
 - ChainOrderPattern
 - ChainPrecedence
 - ChainResponse
 - BooleanExpression
 - BoundaryType
 - AT_LEAST
 - EXACTLY
 - AT_MOST

An instance

ConfigurationA.bpmn

- AtomicProposition[ID=4]: identifier=start,
- AtomicProposition[ID=5]: identifier=finish,
- AtomicProposition[ID=6]: identifier=canEnd
- globally
- canEnd
- universality canEnd
- globally universality canEnd
- globally
- finish
- start
- finish response start
- globally finish response start
- AtomicProposition[ID=44]: identifier=clear
- globally
- ((!(finish)) | clear)
- universality (!((finish)) | clear)
- globally universality (!((finish)) | clear)

Nom	p3
requirement	> Req_3
property	globally universality (!((finish)) clear)
GPL	[(!((!(finish)) clear))]
Description	if the process execution is ended, that is there is no more token left.

Summary

Objective

- Formalization of properties from business requirements
- Traceability between requirement, property and model of the PDP

Approach et solution

- **Dwyer's pattern**
 - Facilitate properties formalization
 - Abstract from different temporal logics
- **Federate modeling spaces**
 - Abstraction of languages and tools
 - Traceability of requirements
- **Generation of properties to be verified by OBP**