# • Expression of properties
# • on BPMN models


- **Valery MONTHE**
  - **ENSTA Bretagne**

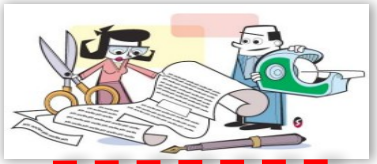
# Plan

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

# Context

- **Requirements**
  - Constraints to be respected/satisfied by a system
  - Written in natural language (subject to ambiguities)
  - Need to be verified
- Question : **Does system meet its requirements ?**

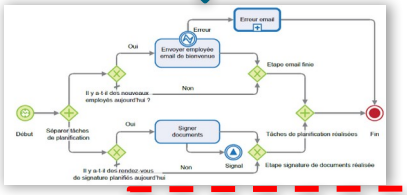
-----??-----

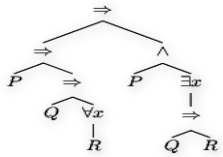


?

System

Requirements





Model

Properties

Verification

# Issue

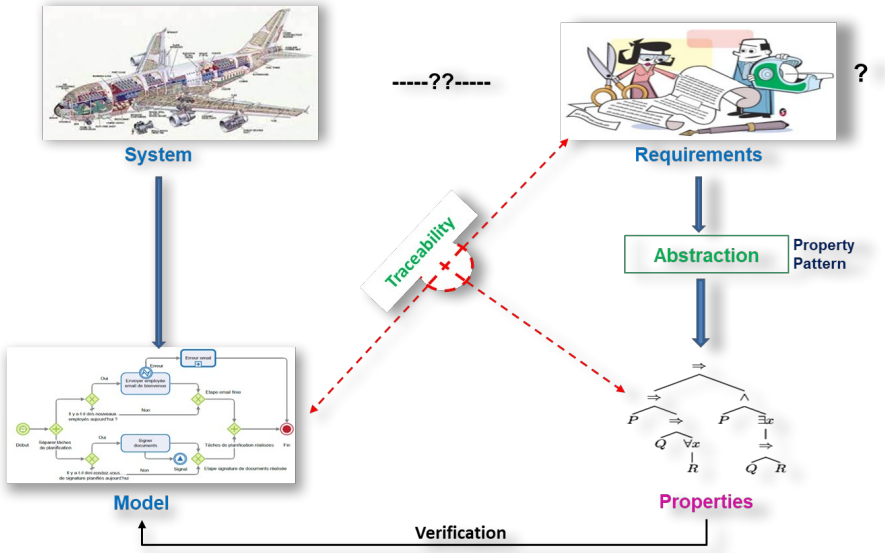
- **An illustrative example**
  - **Requirement**
    - The production of component A leads to the production of component B
    - Each time component A is produced, component B will be
  - **Property in temporal logic**
    - P : component A is produced (**atomic proposition**)
    - S : component B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

❑ **Formal language and temporal logic**
- A formalism far from business experts
- Several temproral 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 temproal 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 -> < > S )** *LTL* => ***Globaly S Responds to P***
    - **AG ( P -> AF ( S ) )** *CTL*

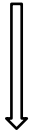
Porté Pattern

- Another example

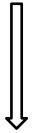
- [ ](Q -> ((P -> (!R U S)) U R) | [ ](P -> (!R U S))) => **After Q until R (S *Responds to P)***

# Dwyer's Pattern

- **Requirement**



- **Abstraction**
  **Dwyer's Pattern**



- **LTL Property**
  (temporal logic)

The production of component A leads to the production of component B

P : component A is produced

S : component B is produced

*Globaly S Responds to P*

Scope Pattern

**[ ] ( P -> < > S )** *LTL*

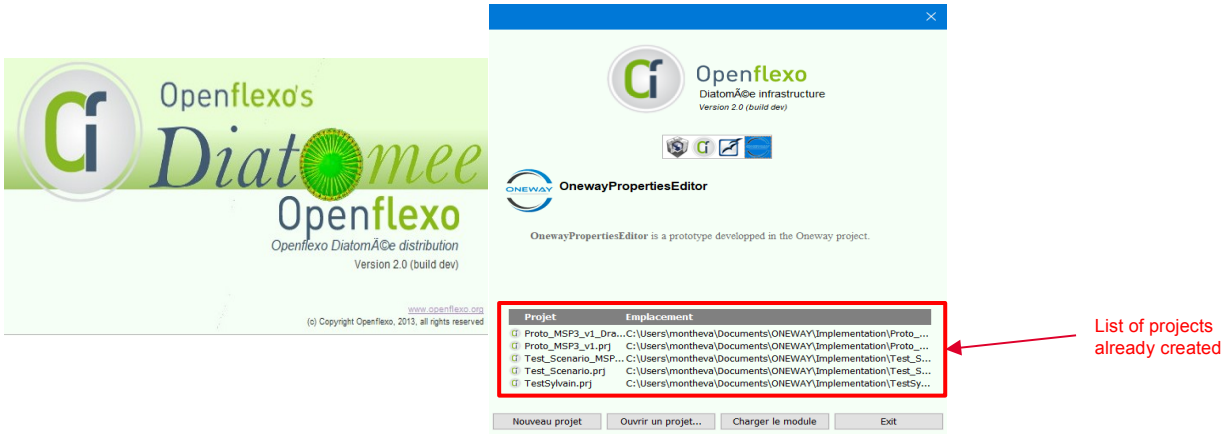
**AG ( P -> AF ( S ) )** *CTL*

# Modeling environment

## Heterogeneous modeling platform : Oneway Properties Editor

An environment that allows you to :

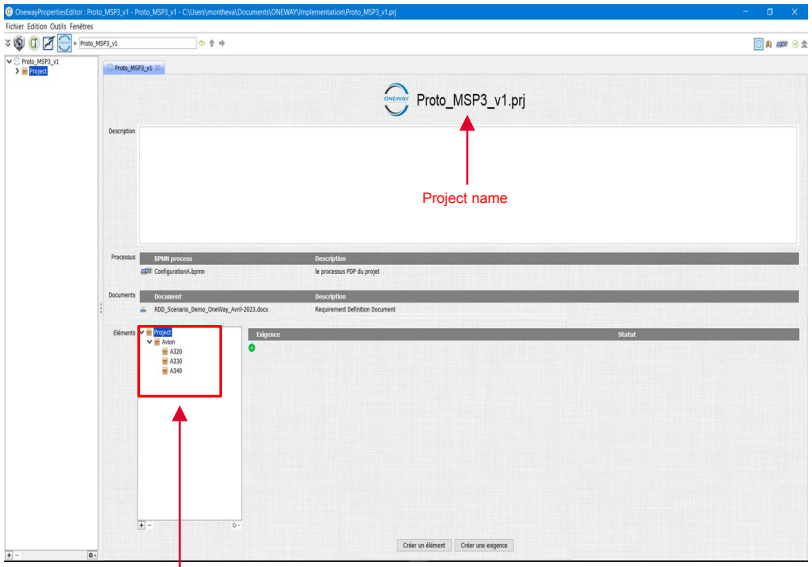
1. Load and read the requirements definition document (Word, Excel, pdf, etc.)
2. Create requirements and link them to document content
3. Load the models of the processes to check
4. Define the properties verifying each requirement, in a language close to the natural
5. Generate the temporal logic formula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

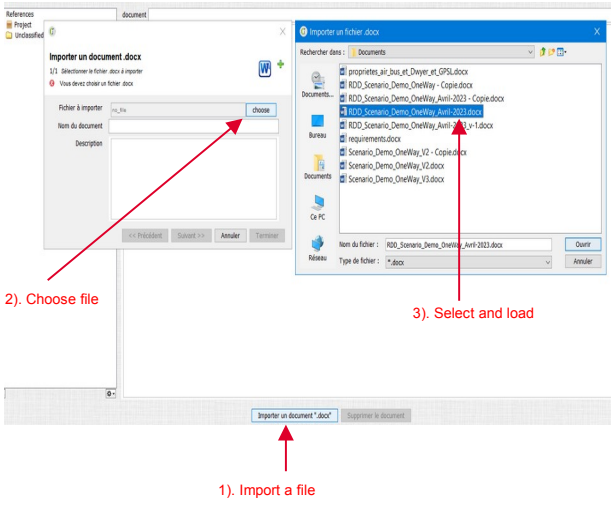


# Usage scenario

## 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

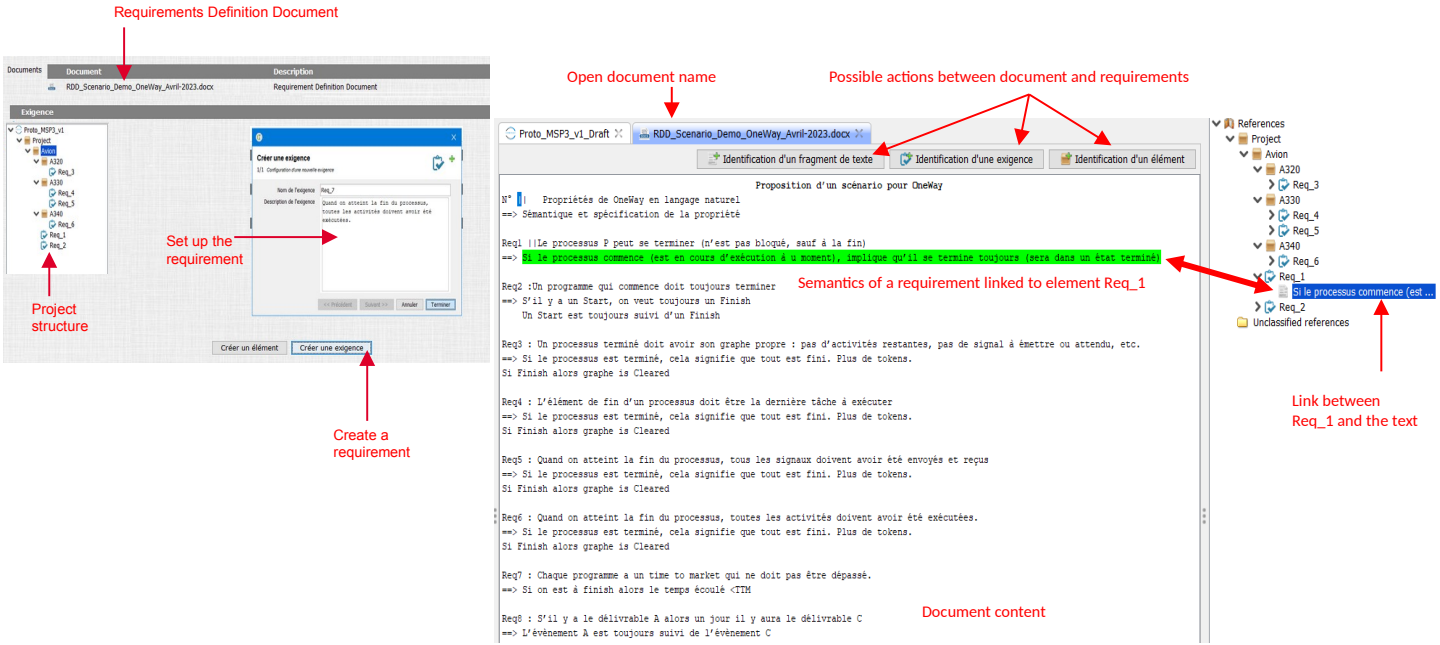


# Usage scenario

## 2. Create requirements and link to the requirements definition document

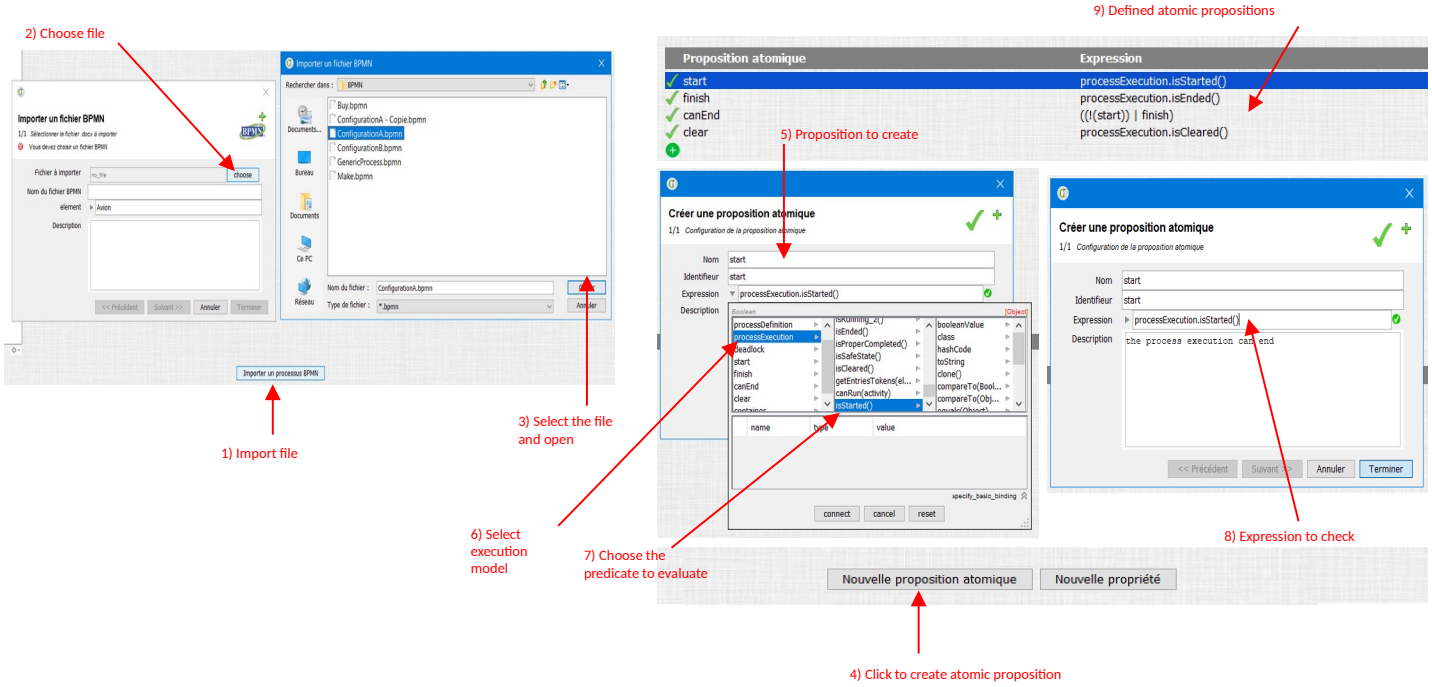
- Instantiation of the requirements contained in the document
- Instantiating relationships between a requirement and its specification contained in the document



# Usage scenario

## 3. Defining predicates: atomic propositions

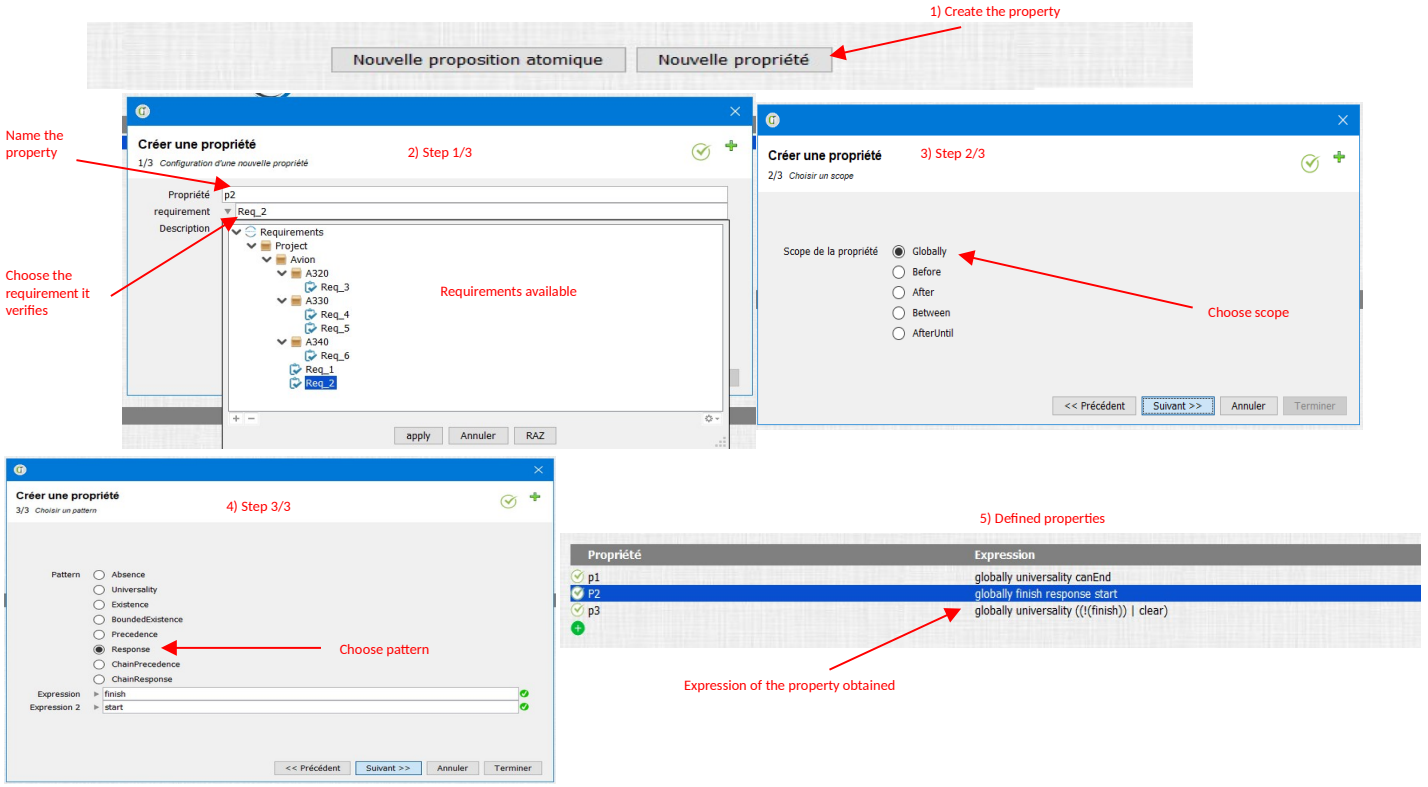
- Load an instance of the PDP (BPMN model)
- Define predicates on the execution model



# Usage scenario

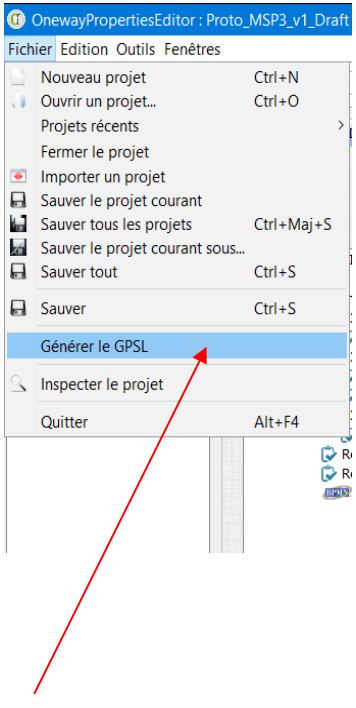
## 4. Define properties to verify

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



# Usage scenario

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



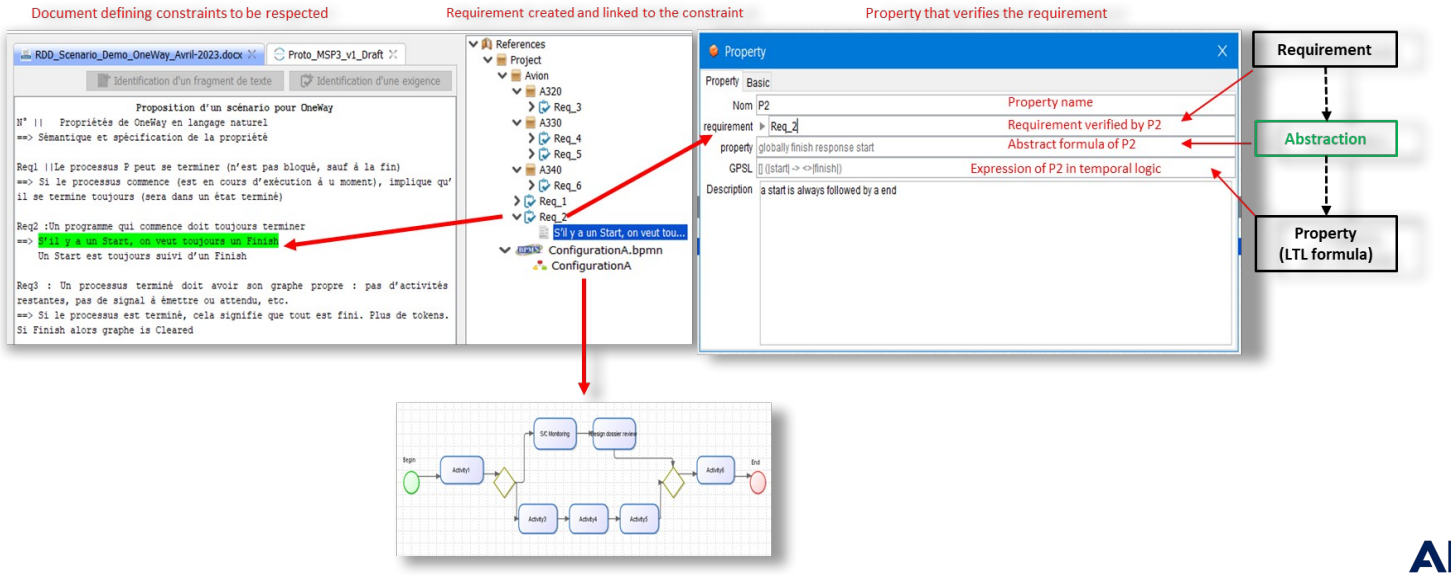

GPSL File Contents

```
1  //-------------------------------------------------------------
2  //-----------------GPSL file-------------------------
3  //-------------------------------------------------------------
4  // this file contains all the properties defined for your projcet
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

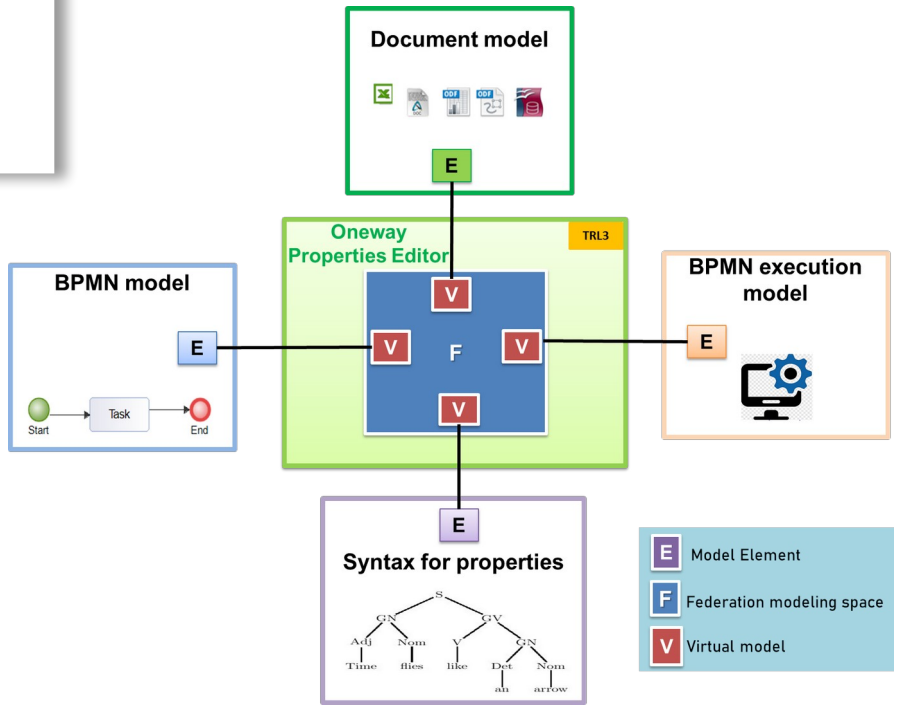
## 6. summarize : end-to-end traceability

- 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 propertuy« ***[ ] (|Start| -> <> |Finish|) »***



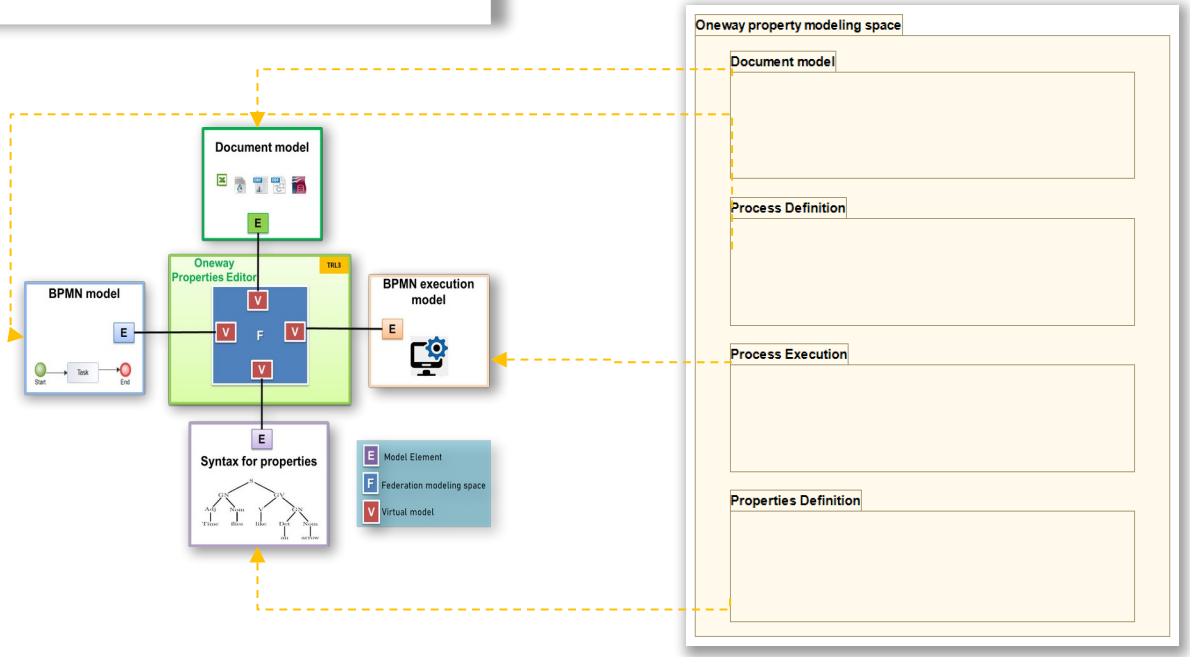
# Conceptualization : Federation

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



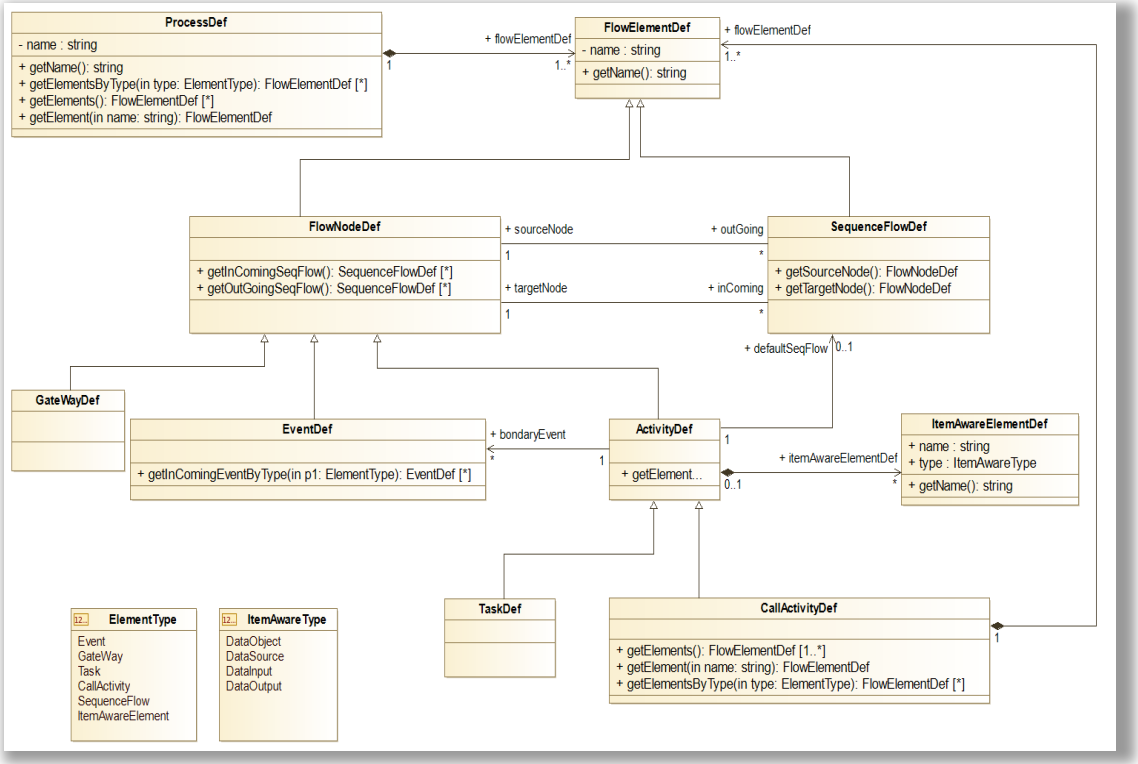
# Conceptualization : Federation

- **A unique space to manipulate heterogeneous models**
  - ✓ BPMN model
  - ✓ BPMN execution model
  - ✓ Predicate language
  - ✓ Syntax for properties definition
  - ✓ Document model



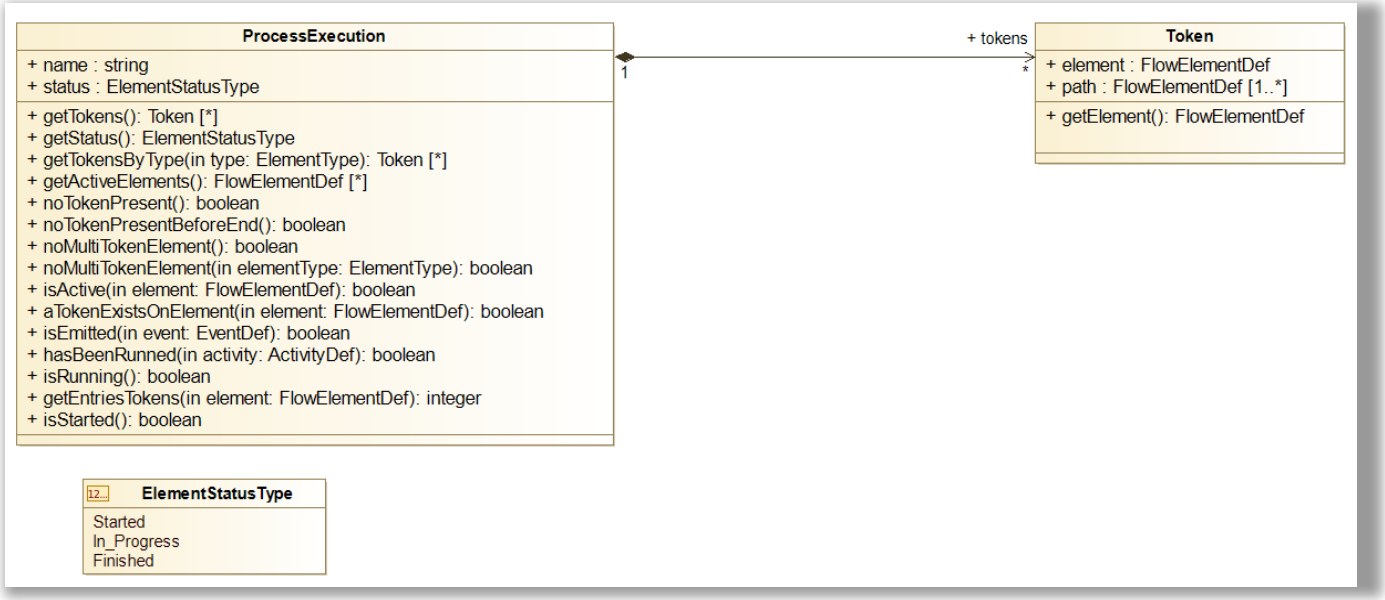
# Conceptualization : Process model

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



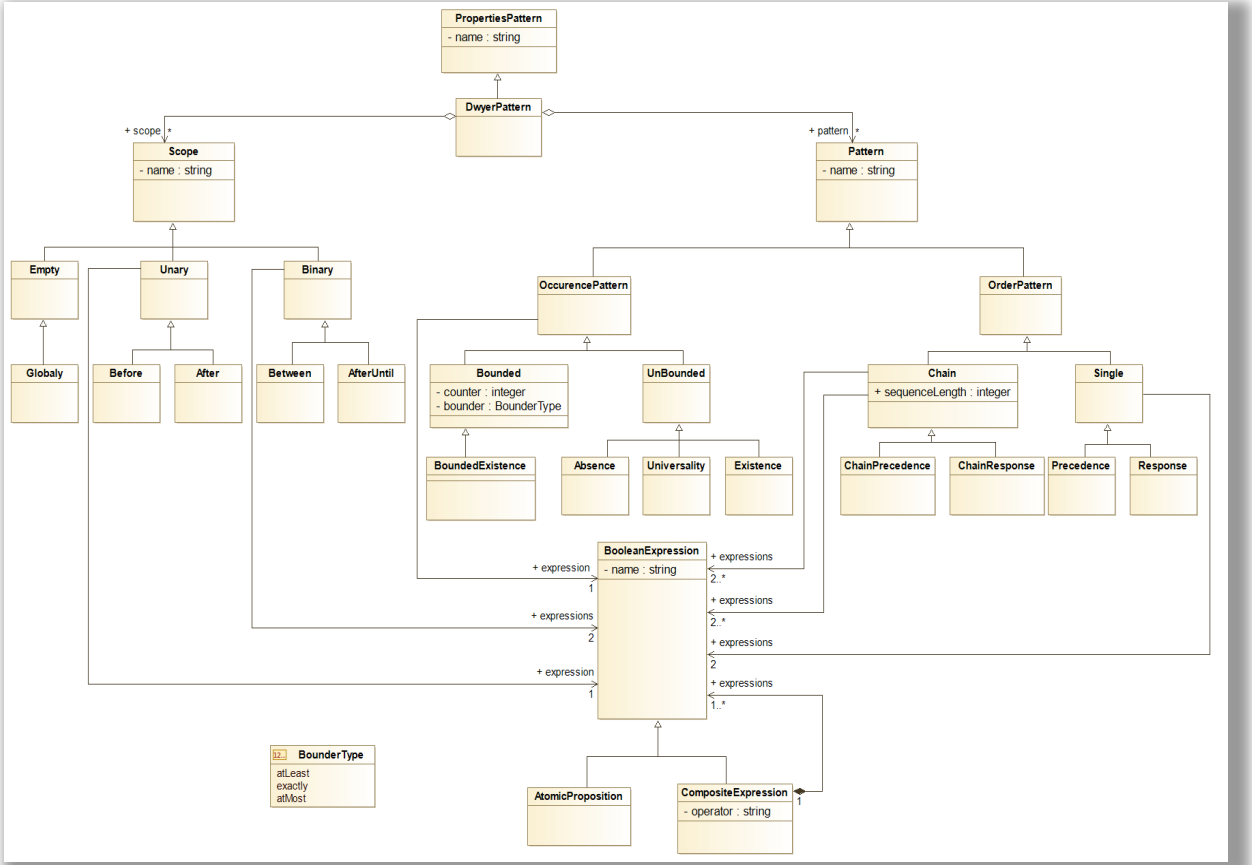
# Conceptualization : Process execution

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



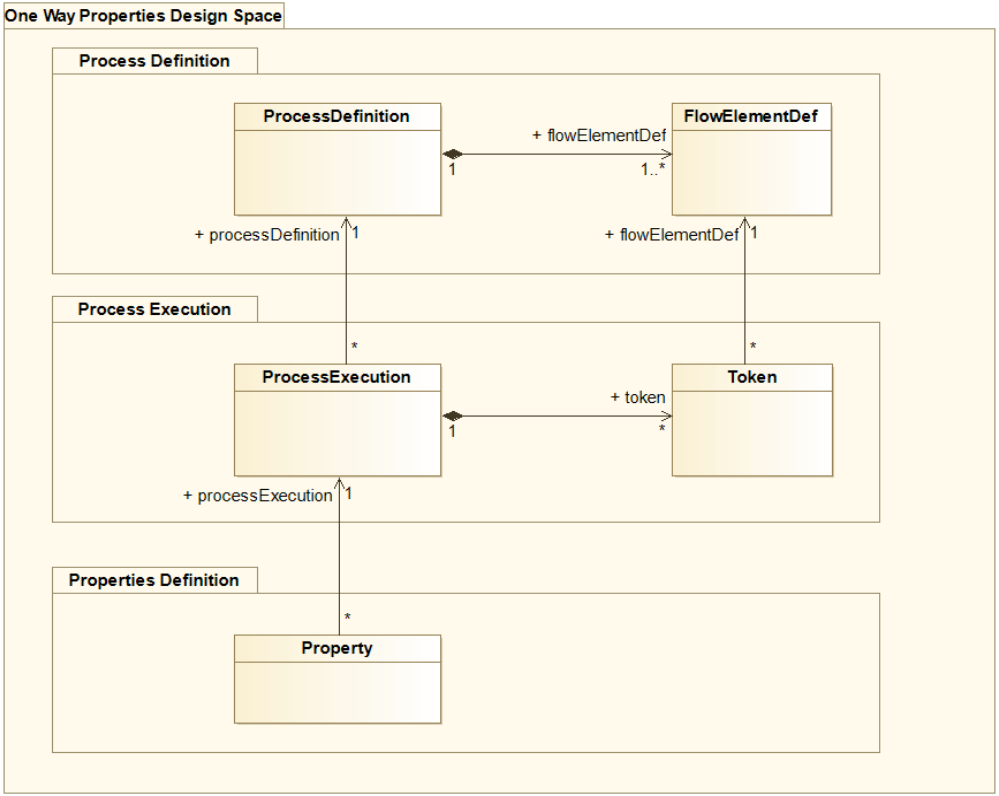
# Conceptualization : Property definition

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



# Conceptualization : Model interaction

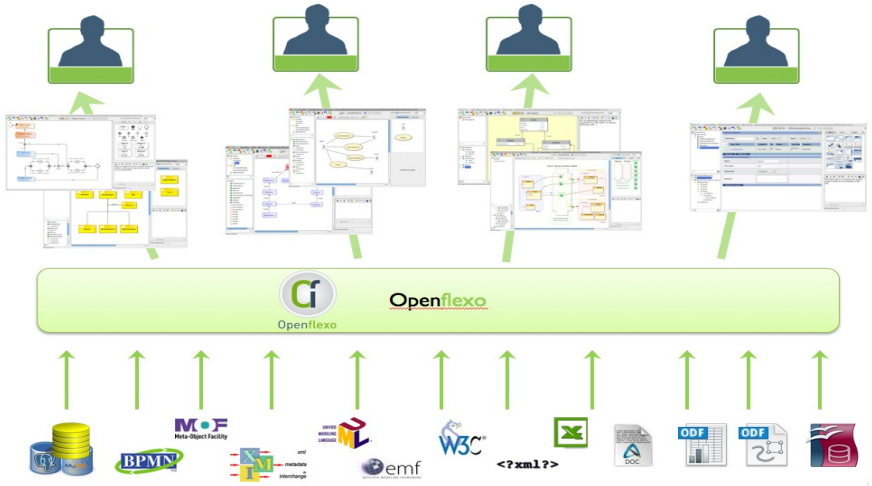
- Properties are set on a process execution
- Execution model manages a set of tokens related to process elements



# Toolchain : Openflexo

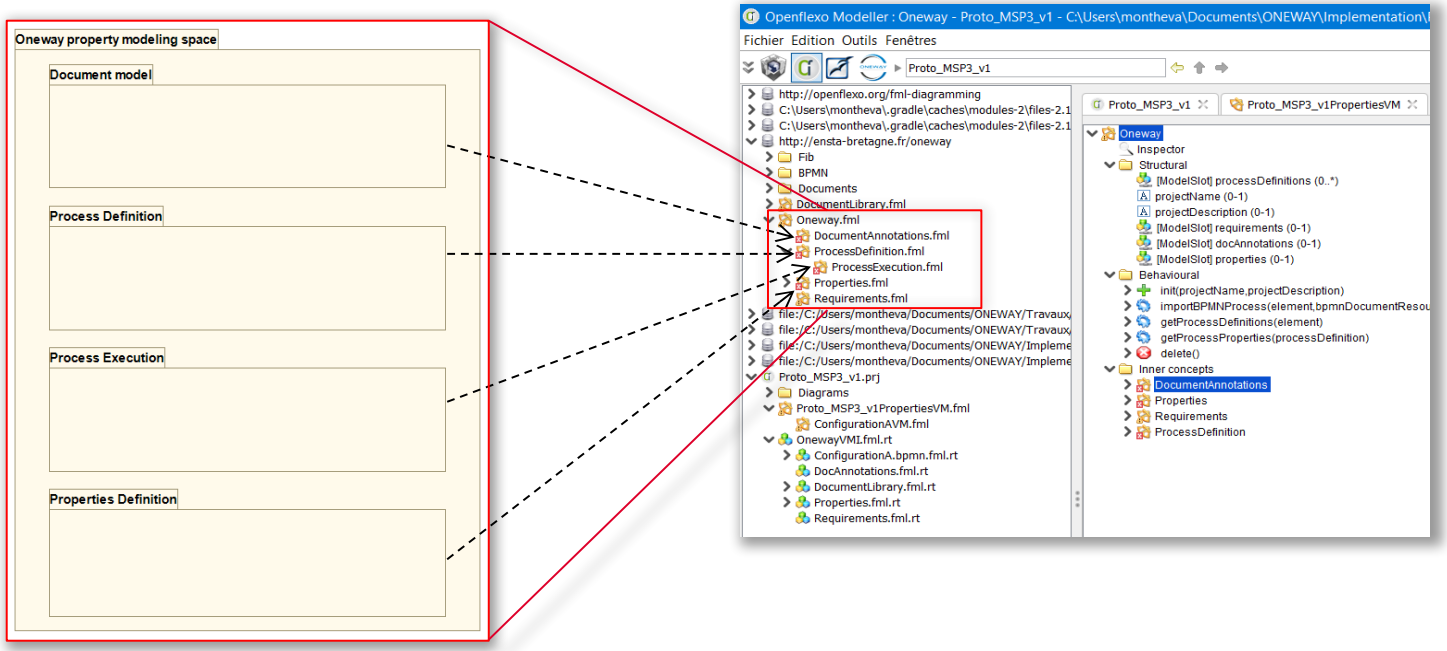
## A model federation framework

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



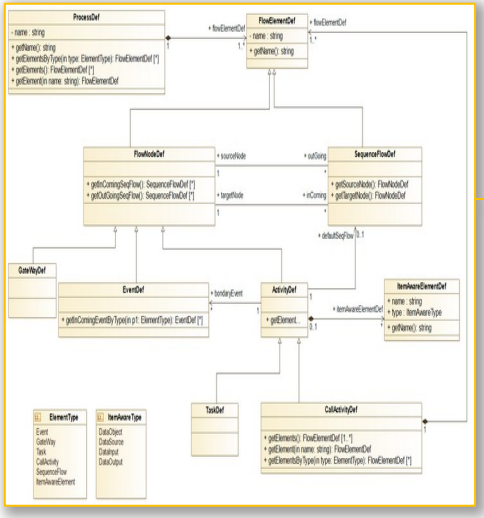
# Implementation : Modelling space

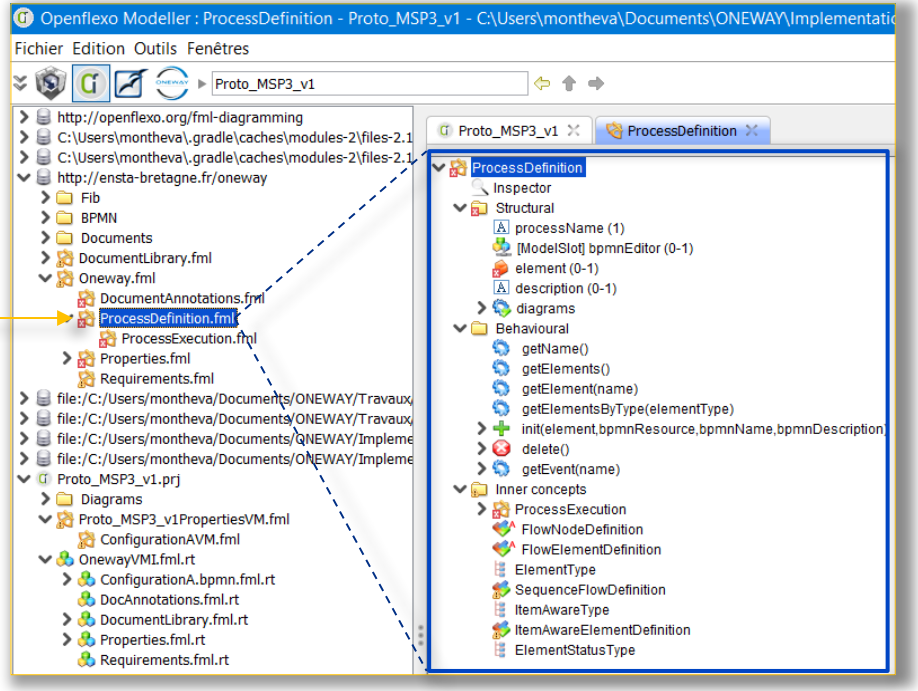
- A modelling space with several subspaces
- Connecting heterogeneous models



# Implementation : Process definition

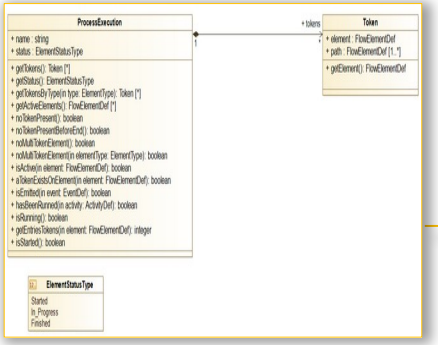
- Implementation of an excerpt of the BPMN metamodel

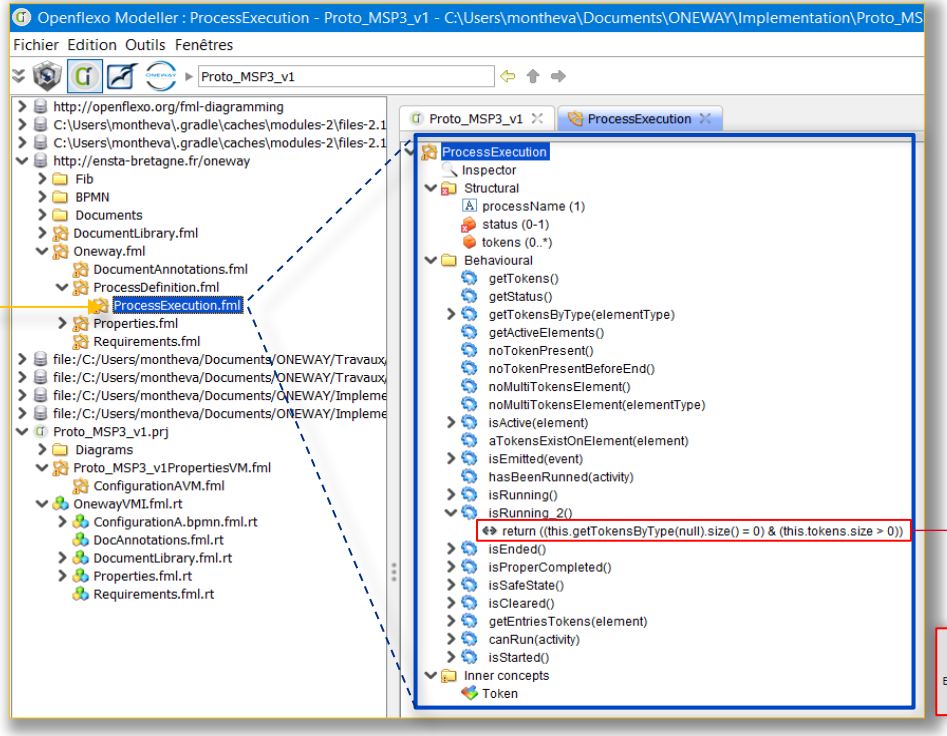




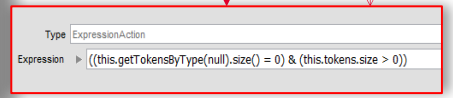
# Implementation : Process execution

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



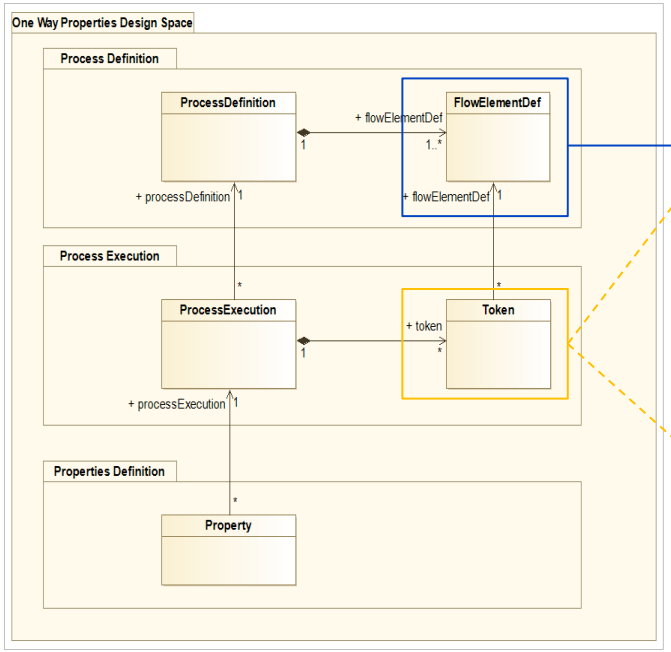



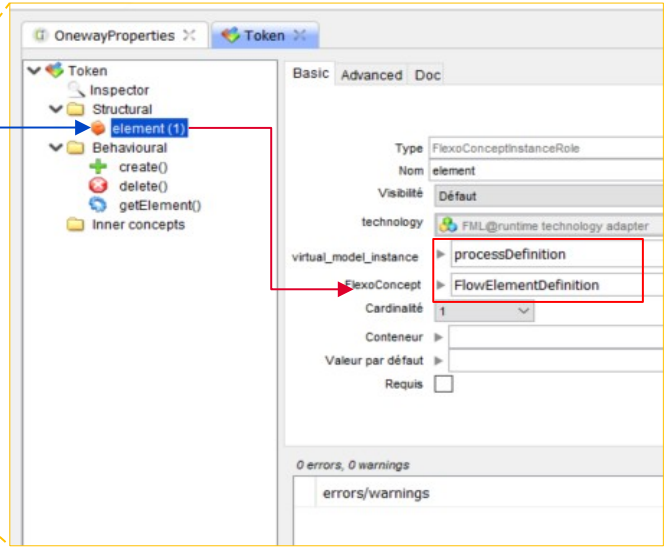

*Evaluable logical expression*



# Implementation : Process execution

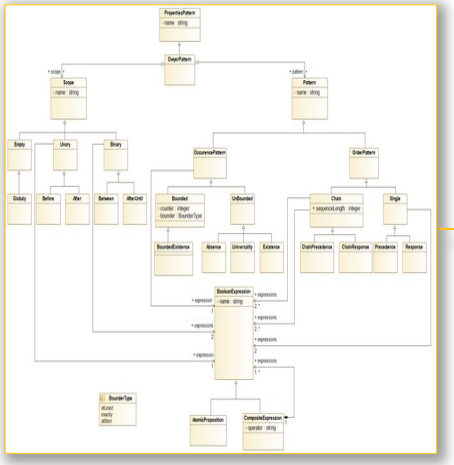
- A token refers to an element of the process definition

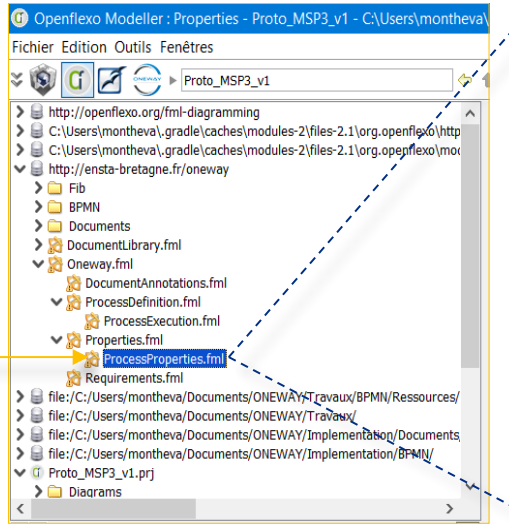


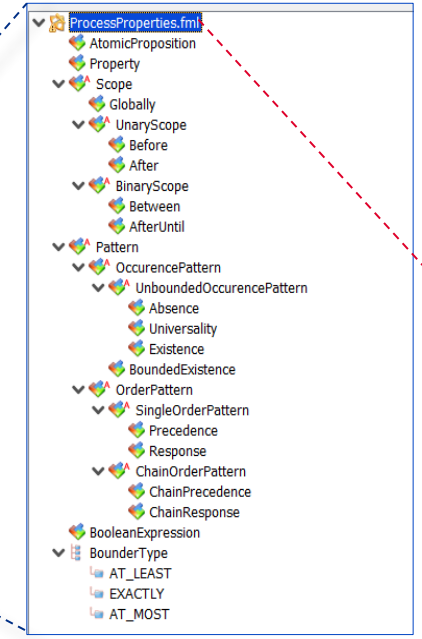



# Implementation : Property definition

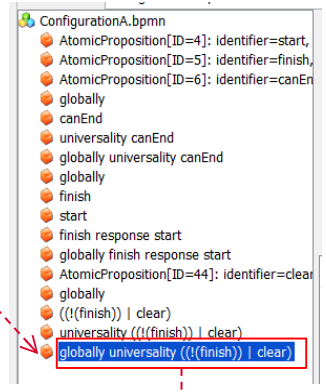
- An abstraction based on Dwyer's patterns
- Designing a Dwyer's pattern metamodel

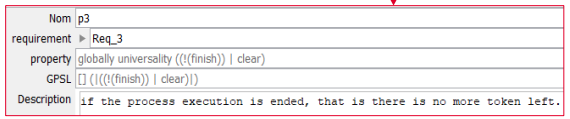





An instance



# 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**