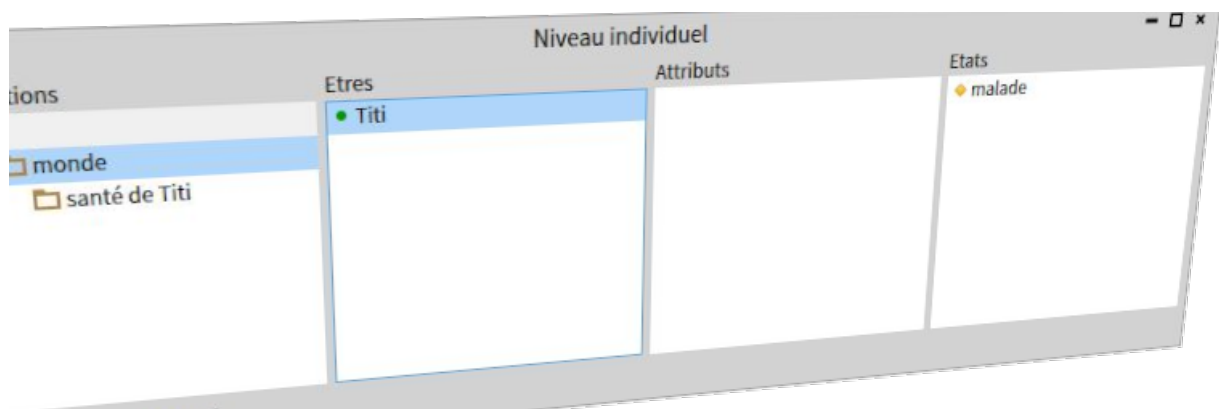


# ICEO par l'exemple

Présentation du langage textuel d'ICEO et de son  
implantation en Smalltalk

Robert Bourgeois



Ce document a été écrit en utilisant l'application TeXstudio

Les graphes présentés dans ce document ont été réalisés avec l'outil graphique Yed de yWorks

# Contents

<b>Avant-propos</b>	<b>3</b>
<b>Installation d'ICEO dans Pharo et exemples</b>	<b>5</b>
Installation d'ICEO . . . . .	5
Services accessibles via l'onglet "ICEO" de la fenêtre principale . . . . .	10
Exemple 2 : sur la composition d'une essence . . . . .	12
Exemple 3 : sur la composition d'une essence à partir d'essences existantes . . . . .	14
Exemple 4 : sur le principe d'héritage des attributs des essences . . . . .	16
Exemple 5 : sur le principe d'identification des essences . . . . .	18
Exemple 5_2 . . . . .	19
Exemple 6 : sur le principe d'identification des êtres . . . . .	21
Exemple 7 : sur la notion de méta essence (essence d'une essence) . . . . .	23
Exemple 8 : sur les notions de situation générique et de situation individuelle . . . . .	27
Exemple 9 : sur la notion de manière d'être essentielle et permanente . . . . .	29
Exemple 10 : sur la notion de manière d'être accidentelle . . . . .	31
Exemple 11 : sur les attributs d'une manière d'être . . . . .	33
Exemple 12 : sur les relations entre manières d'être . . . . .	35
Exemple 13 : sur la subsomption des manières d'être . . . . .	37
Exemple 14 : sur la relation entre états . . . . .	40
Exemple 14_2 . . . . .	42
Exemple 14_3 . . . . .	43
Exemple 14_4 . . . . .	44
Exemple 14_5 . . . . .	46
Exemple 15 : sur la notion d'état dans un état . . . . .	48
Exemple 16 : sur la notion d'être inconnu . . . . .	50
Exemple 17 : sur la notion de contrainte structurelle interne . . . . .	52
Exemple 18 : sur la notion de famille . . . . .	56
Exemple 19 : sur la notion d'action . . . . .	60
Exemple 20 : sur les changements de situation . . . . .	62
Exemple 21 : sur la notion de couleur . . . . .	65
Exemple 22 : les tours de Hanoï revisitées . . . . .	71
<b>Sur l'implantation d'ICEO en Pharo</b>	<b>75</b>



# Avant-propos

Les concepts illustrés ici par des exemples sont présentés dans le document intitulé "Sur la modélisation des êtres et de leurs manières d'être dans certaines situations".

Ce document (ICEO.pdf) est téléchargeable à l'adresse <https://github.com/rodejaphgh/ICEO>.

Le langage textuel d'ICEO est implanté en Smalltalk<sup>1</sup>

Le choix de ce langage est lié à l'élégance de sa syntaxe mais aussi et surtout au fait que de nombreux concepts d'ICEO sont inspirés ou sont ceux de ce langage. La preuve en est que l'implantation d'ICEO en Smalltalk se contente de quelques centaines de lignes de code.

Nous avons utilisé Pharo (version 12) qui est l'une des variantes de Smalltalk dérivées de la version de Smalltalk-80 originelle.

Cette construction n'a quasiment exigé aucune modification de l'environnement de base de Pharo.

Pour étudier ces exemples, nous vous proposons de le faire de manière interactive dans l'environnement intégré de Pharo.

Ceci sera peut-être pour certains l'occasion de découvrir ce merveilleux langage qu'est Smalltalk.

---

<sup>1</sup>Smalltalk est un langage de programmation orienté objet qui a été conçu en 1972 par des génies informatiques : Alan Kay, Dan Ingals, Ted Kaehler et Adele Goldberg au Palo Alto de Xerox. Ce langage qui a inspiré une multitude d'autres langages (tels que Java, C++, Python, ...) fut l'un des tout premiers à disposer d'un environnement de développement intégré complètement graphique.



# Installation d'ICEO dans Pharo et exemples

Pharo est disponible librement en téléchargement sur le site <https://pharo.org/download>

La version utilisée est la version 12 qui peut être installée en utilisant le launcher de Pharo ou en mode standalone (VM + image).

Pour nos lecteurs qui ne connaissent pas Smalltalk, nous conseillons de commencer par la lecture du livre "*Programmation orientée objet avec Smalltalk*" d'Harald Wertz ou du livre "*Pharo By Example*" de Stéphane Ducasse, Gordana Rakic, Sebastian Kaplar et Quentin Ducasse dont une version libre au format pdf peut être téléchargée à l'adresse

<https://books.pharo.org/pharo-by-example9>

## Installation d'ICEO

En supposant Pharo installé, il convient, pour intégrer ICEO dans l'environnement de Pharo, de récupérer les fichiers sources d'ICEO.

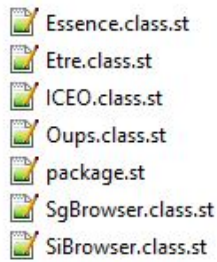
Tous les fichiers sont téléchargeables à l'adresse <https://github.com/rodejaphgh/ICEO>; ils comprennent le code source d'ICEO et le code des exemples présentés dans ce document.

Pour récupérer les fichiers situés sous github, il est conseillé d'utiliser l'outil Metacello intégré dans Pharo. Cet outil est utilisé dans Pharo pour gérer les projets, leurs dépendances et leurs métadonnées.

Ceci peut être fait en ouvrant une fenêtre de type Playground et en évaluant le code :

```
Metacello new baseline: 'ICEO';  
repository: 'github://rodejaphgh/ICEO:main/src';  
load.
```

Ces instructions vont récupérer localement, dans un sous-dossier nommé "pharo-local/iceberg/rodejaphgh/ICEO/src/ICEO" situé dans le dossier où est installé Pharo, tous les fichiers sources situés dans la repository github nommée ICEO :

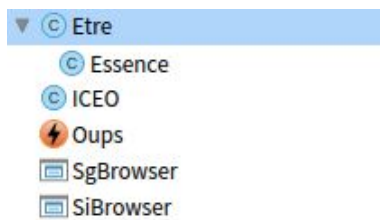


La dernière instruction " load " installe ICEO dans l'environnement de Pharo.

Au bout de quelques instants la barre de menu principale affiche un nouvel onglet nommé ICEO :



Dans un Browser, la catégorie ICEO s'affiche avec en particulier les classes Etre, Essence, et ICEO :

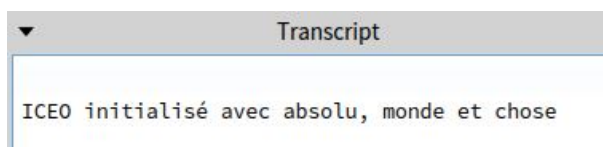


La classe ICEO est l'interprète du langage textuel d'ICEO.

Une instance de cette classe est créée lors de l'installation, nommée "iceo" (en minuscules).

Avant de commencer l'étude d'un premier exemple, il est impératif d'exécuter l'instruction " ICEO start ".

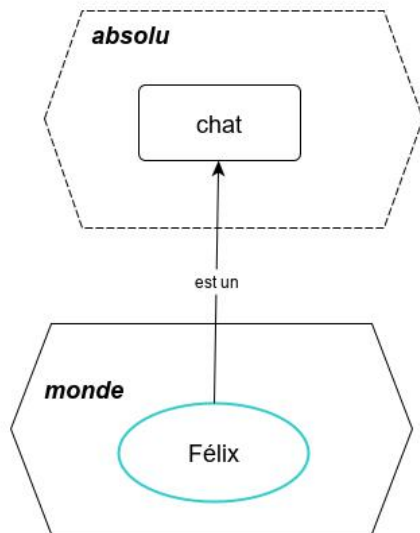
Dans une fenêtre Transcript, le texte suivant s'affiche alors :



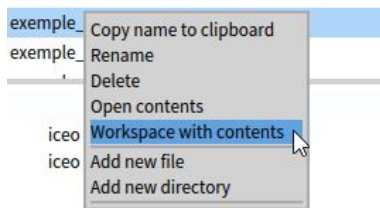
A ce stade, il est conseillé de sauvegarder votre image. Pour retrouver Pharo dans l'état actuel lors du prochain lancement, il suffira ainsi de choisir cette image.

Intéressons-nous maintenant au premier exemple, qui correspond au graphe suivant :





Dans une fenêtre File Browser sélectionner le fichier "exemple\_1.text" et l'option Workspace with contents :



Une fenêtre de type Playground s'affiche avec le code de l'exemple :



Dans le texte de l'exemple, vous pouvez sélectionner et interpréter les instructions (ou expressions) une par une ou en bloc, en utilisant l'option "Do it" offerte avec le bouton droit de la souris.

Pour l'interprétation d'un bloc d'instructions, il faut que chacune se termine par un point.

Si le premier objet de plusieurs instructions consécutives est le même, il est possible de les écrire en cascade, séparées par un point-virgule.

Exemple :

```
ico definition: #chat; soit: #Félix essence: chat.
```

Une longue instruction peut s'étendre sur plusieurs lignes.

Chaque exemple peut être exécuté en totalité en une seule passe en utilisant une combi-

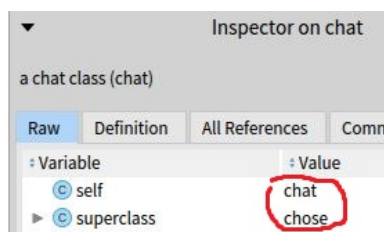
raison des touches Ctrl a (pour sélectionner l'ensemble du texte) suivie d'un DoIt ou de Ctrl d<sup>1</sup>

Un conseil : ayez toujours une fenêtre de type Transcript ouverte lors de l'exécution d'un bloc d'instructions. En cas d'erreur, l'instruction fautive y sera affichée.

La première instruction du premier exemple définit l'essence chat dans la situation générique absolu, subsumée par l'essence chose.

Rappelons que dans ICEO le nom commun des essences s'écrit obligatoirement (comme en langage naturel) avec une minuscule et que le nom propre des êtres s'écrit habituellement avec une majuscule.

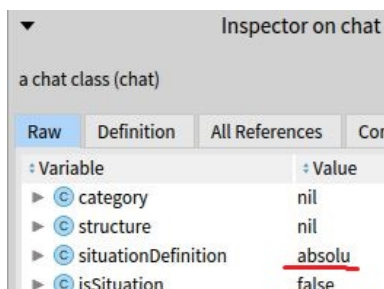
Sélectionnez le mot chat et utilisez l'option "Inspect it" offerte avec le bouton droit de la souris :



Vous voyez que l'essence chat a été créée sous la forme d'une classe Smalltalk. Les différents attributs qui sont listés seront expliqués au moment opportun.

L'attribut "superclass" montre que le genus de l'essence chat est l'essence chose.

L'attribut "situationDefinition" montre que l'essence chat est définie dans l'absolu :




L'accès à l'essence chat pourrait se faire avec l'expression " `absolu get: #chat` " mais ce n'est pas nécessaire car les essences définies dans l'absolu sont accessibles directement par leur nom.

La deuxième instruction de l'exemple crée l'être nommé Félix comme instance de chat dans le monde.

L'expression "`monde get: #Félix` " montre que Félix est un chat présent dans le monde :

---

<sup>1</sup>ou en cliquant sur le petit bouton , si vous souhaitez exécuter l'exemple en totalité et ouvrir un inspecteur sur le résultat de la dernière instruction exécutée

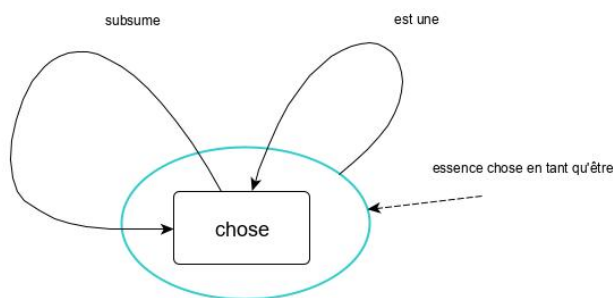
Inspector on a chat		Inspector on an absolu	
a chat		an absolu	
Raw	Breakpoints	Raw	Breakpoints
Meta	Meta	Raw	Breakpoints
Meta	Meta	Meta	Meta
Variable	Value	Variable	Value
self	a chat	self	an absolu
structure	nil	{ } structure	an OrderedCollecti
situationDefinition	an absolu	situationDefinition	nil
isSituation	false	isSituation	true
isEtat	false	isEtat	false
etats	nil	etats	nil
etant	nil	etant	nil
nom	Félix	nom	monde
id	nil	id	nil

Noter que l'accès à l'être nommé Félix situé dans le monde exige de passer par l'expression "monde get: #Félix ", car Félix ne se situe pas dans l'absolu comme l'essence chat mais dans le monde.

L'essence chose définie dans l'absolu est son propre genus et, en tant qu'être, est instance d'elle-même (elle est sa propre méta-essence).

Ainsi les expressions " chose getGenus " et " chose getEssence " retournent chose.

Ceci est conforme au schéma :



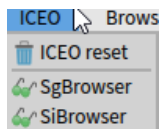
Par défaut, l'essence d'une essence (sa méta-essence) est l'essence chose.

Ainsi, l'expression " chat getEssence " retourne chose.

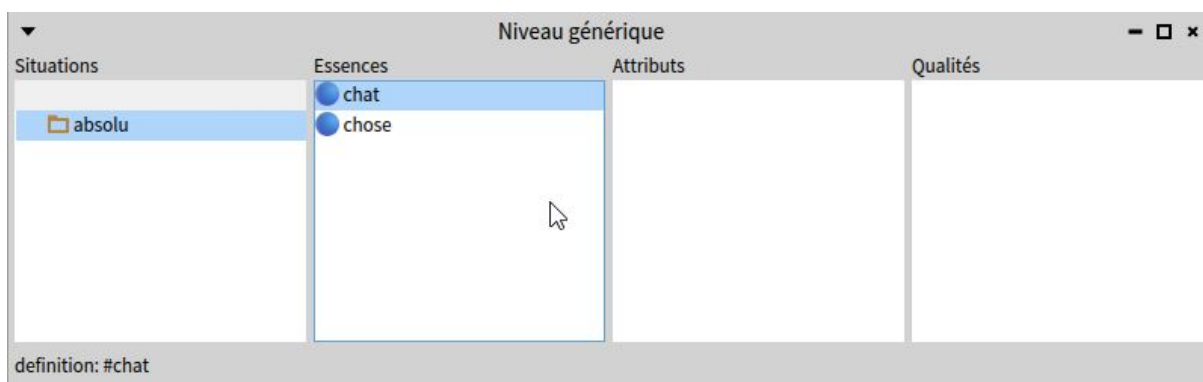
Vous êtes maintenant parés pour étudier d'autres exemples.

## Services accessibles via l'onglet "ICEO" de la fenêtre principale

Ces services sont également accessibles avec le bouton gauche de la souris dans la fenêtre principale.



L'option "SgBrowser" ouvre une fenêtre sur les situations génériques et les essences définies dans l'absolu

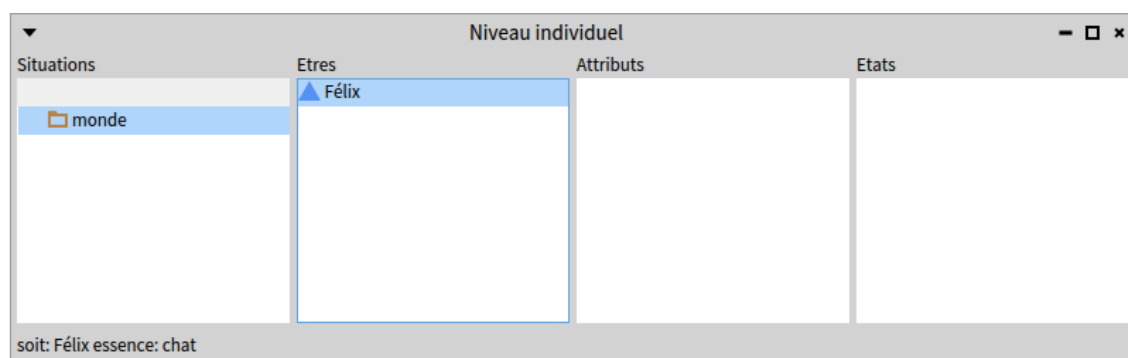


Le nom des essences est précédé d'un petit cercle bleu.

Un double-click sur un item (par exemple l'essence chat) ouvre une fenêtre permettant d'inspecter celle-ci.

Le label affiché en bas de la fenêtre rappelle l'instruction qui a donné naissance à l'entité sélectionnée.

L'option "SiBrowser" ouvre une fenêtre sur les situations individuelles et les êtres définis dans le monde :



Le nom des êtres est précédé d'un petit triangle bleu.

L'option "ICEO reset" réinitialise le contenu de absolu et de monde. Ceci évite qu'ICEO ne conserve la mémoire des exemples précédemment étudiés.

En principe, cette instruction n'est pas nécessaire ici, car il s'agit de notre premier exemple.

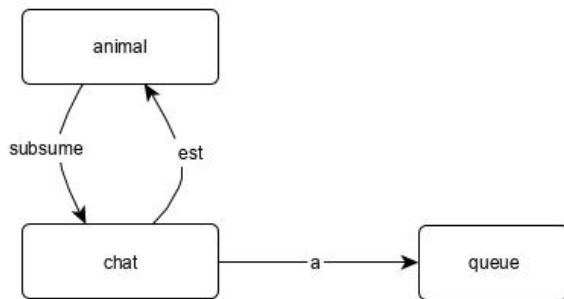
Par contre, il pourra être nécessaire de l'exécuter avant de démarrer l'étude d'un autre exemple, pour éviter des interférences avec le précédent.

Rappelons que dans ICEO :

- deux essences de même nom ne peuvent être définies dans la même situation.
- deux êtres de même nom ne peuvent coexister dans une situation, sauf s'ils peuvent être distingués par l'un de leurs états (par exemple, deux êtres nommés Pierre dont l'un est fils et l'autre père dans une même famille).

## Exemple 2 : sur la composition d'une essence

Cet exemple correspond au graphe suivant :



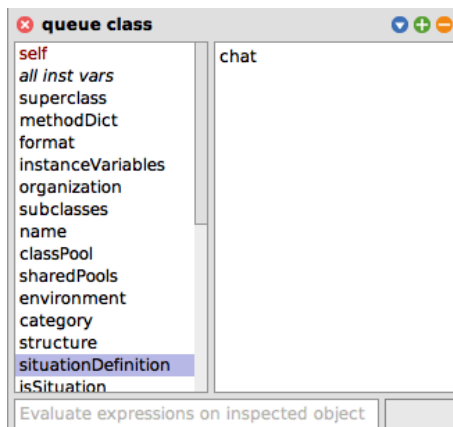
chat est animal qui a queue (chat =  $\oplus$  (animal, { queue}))

L'essence chat subsumée par animal possède un attribut propre nommé queue :

```
iceo definition: #animal.  
iceo definition: #chat genus: animal.  
iceo definitionAttribut: #queue de: chat.
```

Les essences animal et chat ont été définies dans l'absolu, tandis que l'essence chat constitue la situation de définition de sa queue, ce qui peut être vérifié en faisant un "Inspect it" sur l'expression donnée en commentaire entre doubles quotes :

"chat getEssenceAttribut: #queue " :



Ceci est confirmé dans un SgBrowser :

Niveau générique

Situations

absolu

Essences

animal

chat

chose

Attributs

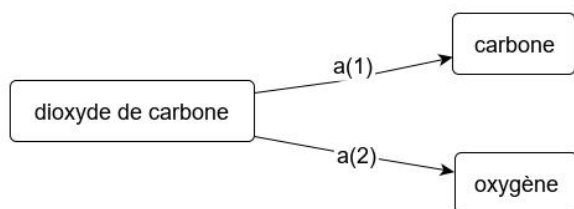
queue de chat

Qualités

definition: #chat genus: #animal

### Exemple 3 : sur la composition d'une essence à partir d'essences existantes

Cet exemple correspond au graphe suivant :

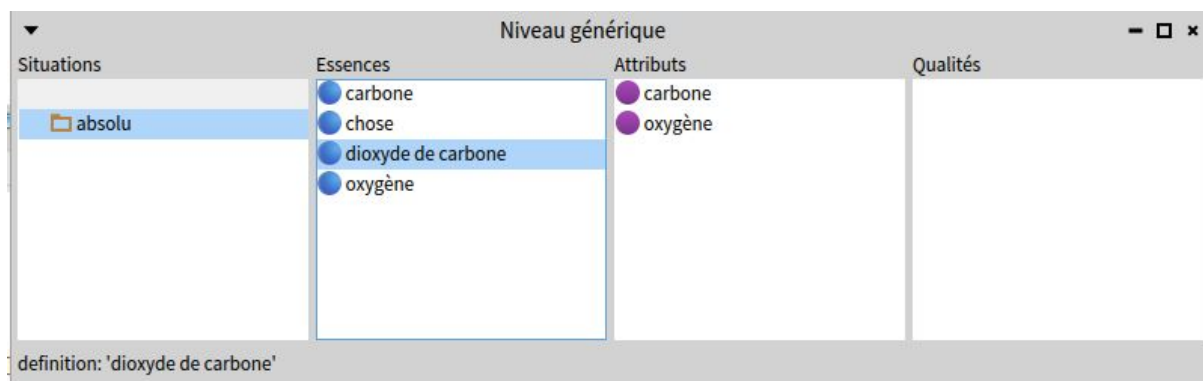


L'essence 'dioxyde de carbone' comporte un atome de carbone et deux atomes d'oxygène qui ne lui sont pas propres :

```
iceo definition: #oxygène.  
iceo definition: #carbone.  
iceo definition: 'dioxyde de carbone'.  
(absolu get: 'dioxyde de carbone' referenceEssence: oxygène cardinalite: 2.  
(absolu get: 'dioxyde de carbone' referenceEssence: carbone cardinalite: 1.  
iceo soit: 'une molécule de dioxyde de carbone' essence: (absolu get: '  
dioxyde de carbone' ).
```

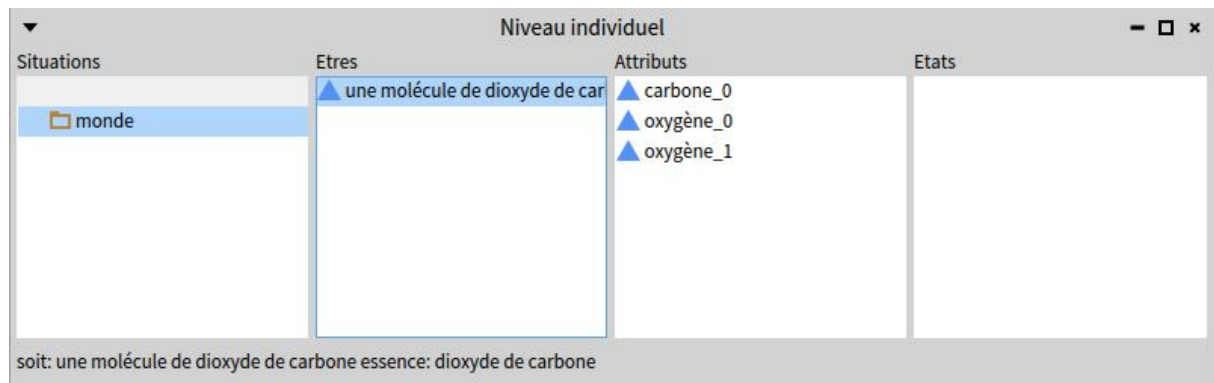
La quatrième instruction indique que l'essence 'dioxyde de carbone' réfère l'essence existante oxygène (définie ici dans l'absolu) avec une cardinalité de 2.

Les attributs carbone et oxygène sont affichés avec un petit cercle violet dans un SgBrowser pour indiquer qu'il ne s'agit pas d'attributs propres de dioxyde de carbone :



Dans un SiBrowser nous voyons que les êtres attributs de 'une molécule de dioxyde de carbone' ont été créés avec la cardinalité définie au niveau de son essence :

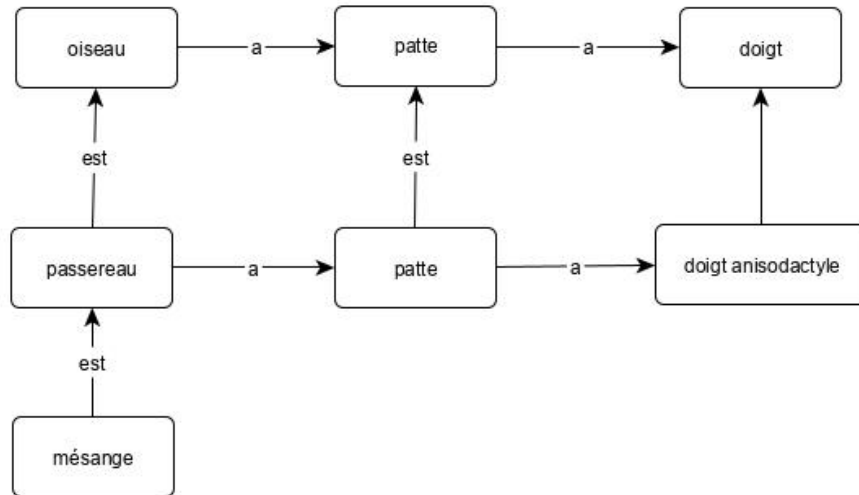




Comme le dioxyde de carbone est un individu, son instanciation a entraîné la création de ses attributs. Le nom des attributs a été généré automatiquement en tenant compte du nom de leur essence.

## Exemple 4 : sur le principe d'héritage des attributs des essences

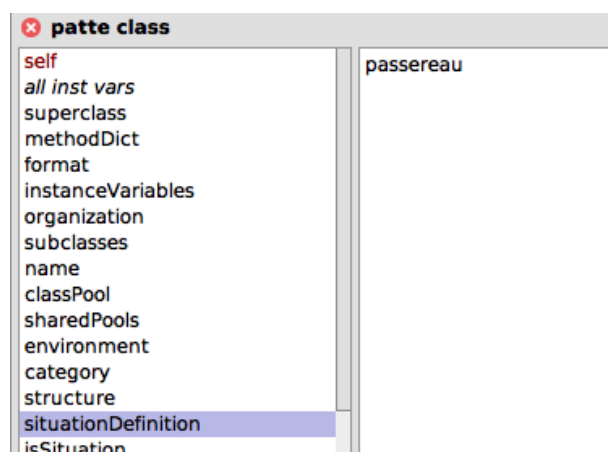
Considérons le graphe suivant :



```

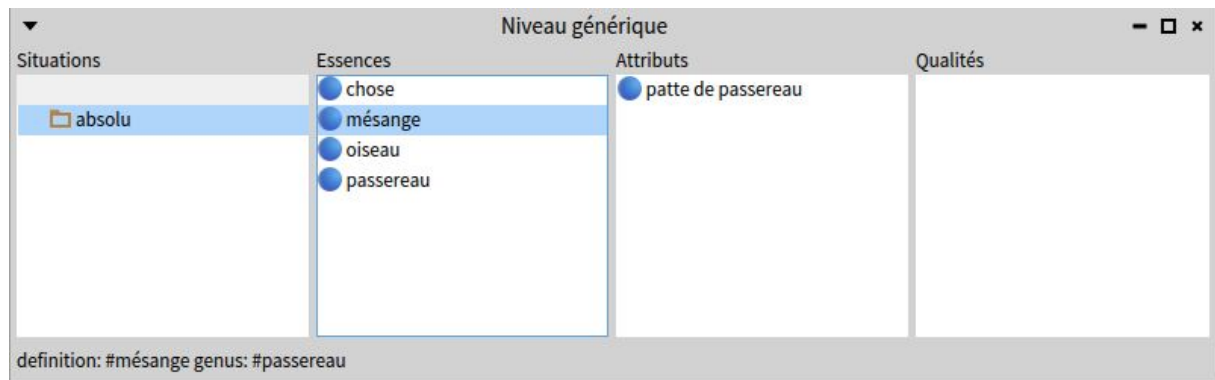
iceo definition: #oiseau.
iceo definition: #passereau genus: oiseau.
iceo definition: #mésange genus: passereau.
iceo definitionAttribut: #patte de: oiseau.
iceo definitionAttribut: #doigt de: (oiseau getEssenceAttribut: #patte).
iceo definitionAttribut: #patte de: passereau
genus: (oiseau getEssenceAttribut: #patte).
iceo definitionAttribut: 'doigt anisodactyle' de: (passereau
    getEssenceAttribut: #patte) genus: ((oiseau getEssenceAttribut: #patte)
    getEssenceAttribut: #doigt) .
  
```

L'inspection de " mésange getEssenceAttribut: #patte " donne :



Il s'agit donc de l'attribut patte de passereau

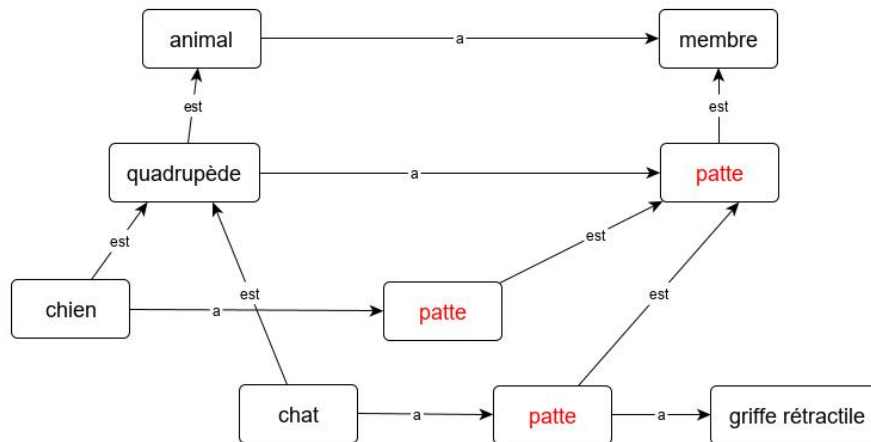
Ceci est confirmé dans un SgBrowser :



L'essence attribut "doigt anisodactyle" de patte de mésange est héritée de patte de passereau, ce qui confirmé par un "Inspect it" de " (mésange getEssenceAttribut: #patte) getEssenceAttribut: 'doigt anisodactyle' "

## Exemple 5 : sur le principe d'identification des essences

Considérons le graphe suivant :



```
iceo definition: #animal.
iceo definition: #quadrupède genus: animal.
iceo definition: #chien genus: quadrupède.
iceo definition: #chat genus: quadrupède.
iceo definitionAttribut: #membre de: animal.
iceo definitionAttribut: #patte de: quadrupède genus: (animal
    getEssenceAttribut: #membre).
iceo definitionAttribut: #patte de: chien genus: (quadrupède
    getEssenceAttribut: #patte).
iceo definitionAttribut: #patte de: chat genus: (quadrupède
    getEssenceAttribut: #patte).
iceo definitionAttribut: 'griffe rétractile' de: (chat getEssenceAttribut:
    #patte).
```

On voit dans cet exemple que diverses essences peuvent avoir le même nom dans des situations différentes.

Du fait que différentes essences peuvent avoir le même nom, il est possible de demander (pour nous, lecteur humain) la génération d'un nom propre à chaque essence, pour vérifier par exemple que l'essence patte de quadrupède est différente de celle de chat et de chien.

Ainsi, un "Print it" des expressions suivantes :

```
" 'patte de quadrupède : ', (quadrupède getEssenceAttribut: #patte) getId "
```

```
" 'patte de chien : ', (chien getEssenceAttribut: #patte) getId "
```

```
" 'patte de chat : ', (chat getEssenceAttribut: #patte) getId "
```

donne :

```
patte de quadrupède : patte_0
```

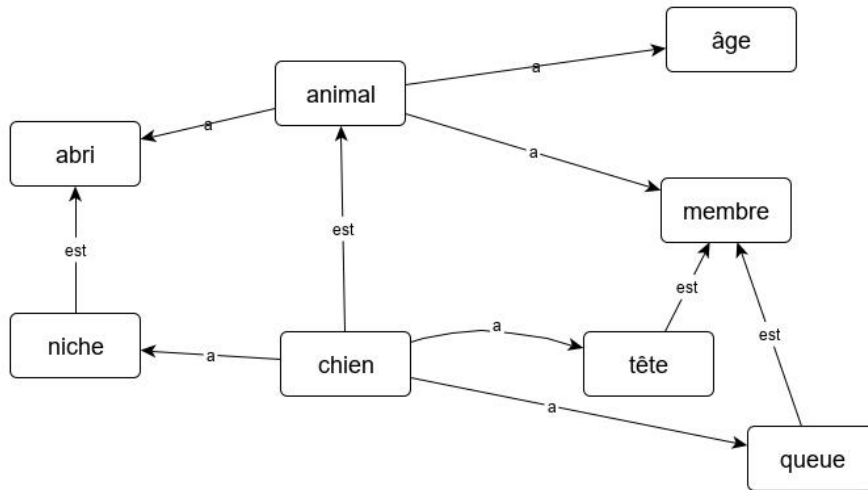
```
patte de chien : patte_1
```

```
patte de chat : patte_2
```

## Exemple 5\_2

L'identification d'une essence peut aussi se faire en utilisant le nom de son genus.

Considérons le graphe suivant :



```
iceo definition: #animal.
iceo definitionAttribut: #âge de: animal.
iceo definitionAttribut: #membre de: animal.
iceo definitionAttribut: #abri de: animal.
iceo definition: #chien genus: animal.
iceo definitionAttribut: #niche de: chien genus: (animal getEssenceAttribut
: #abri).
iceo definitionAttribut: #tête de: chien genus: (animal getEssenceAttribut:
#membre).
iceo definitionAttribut: #queue de: chien genus: (animal getEssenceAttribut
: #membre).
```

L'expression " `chien getEssenceAttribut: #âge` " donne : âge

Il s'agit de l'attribut âge hérité de l'essence animal.

L'expression " `chien getEssenceAttribut: #abri` " donne: niche

et l'expression " `chien getEssencesAttributs: #membre` " donne : an OrderedCollection(queue tête)

Ce qui se vérifie dans les fenêtres suivantes :

Niveau générique

Situations	Essences	Attributs	Qualités
<div>absolu</div>	<div>animal</div> <div>chien</div> <div>chose</div>	<div>abri de animal</div> <div>membre de animal</div> <div>âge de animal</div>	

definition: #animal

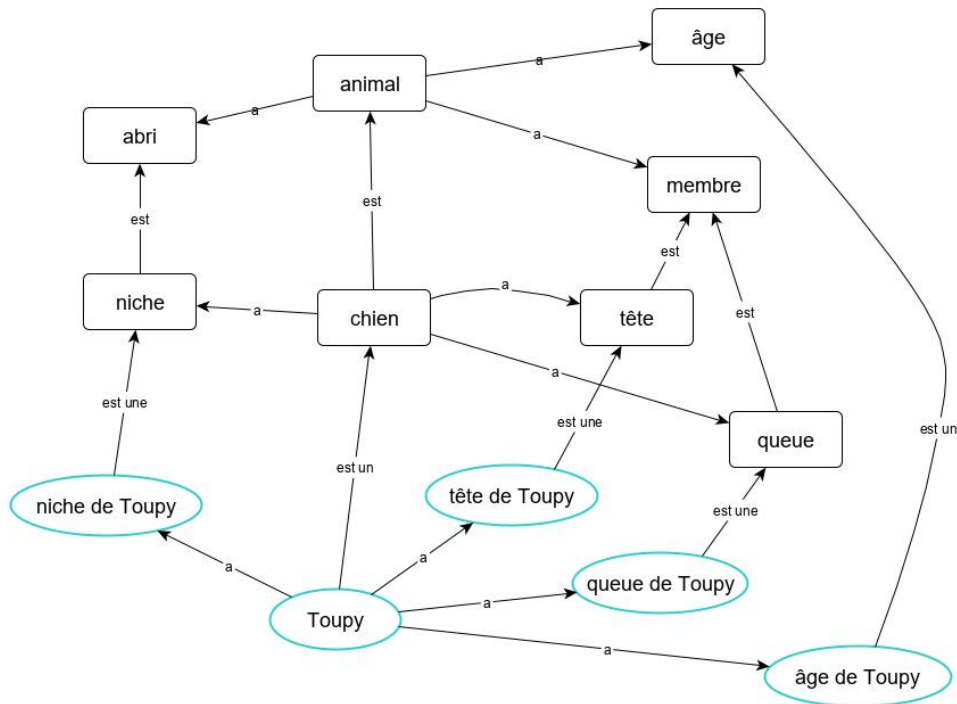
Niveau générique

Situations	Essences	Attributs	Qualités
<div>absolu</div>	<div>animal</div> <div>chien</div> <div>chose</div>	<div>niche de chien</div> <div>queue de chien</div> <div>tête de chien</div> <div>âge de animal</div>	

definition: #chien genus: #animal

## Exemple 6 : sur le principe d'identification des êtres

Considérons le graphe suivant :



```

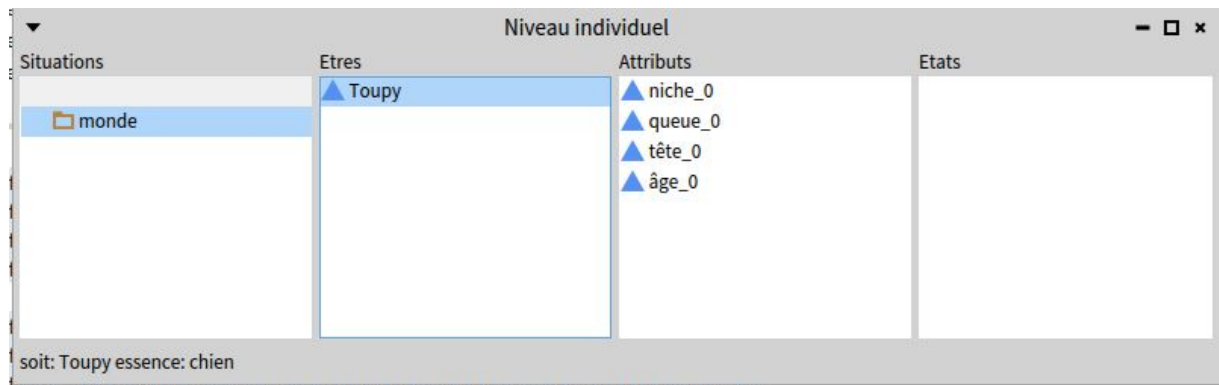
iceo definition: #animal.
iceo definitionAttribut: #âge de: animal cardinalite: 1.
iceo definitionAttribut: #membre de: animal.
iceo definitionAttribut: #abri de: animal.
iceo definition: #chien genus: animal.
iceo definitionAttribut: #niche de: chien genus: (animal getEssenceAttribut
: #abri) cardinalite: 1.
iceo definitionAttribut: #tête de: chien genus: (animal getEssenceAttribut:
#membre) cardinalite: 1.
iceo definitionAttribut: #queue de: chien genus: (animal getEssenceAttribut
: #membre) cardinalite: 1.
iceo soit: #Toupou essence: chien.

```

L'expression " (monde get: #Toupou) getEtresAttributs " donne : an OrderedCollection(âge\_0 niche\_0 tête\_0 queue\_0)

Les attributs de Toupou ont été créés automatiquement car il s'agit par défaut d'un individu.

Leur nom a été créé par un générateur de symboles en se basant sur le nom de leur essence.

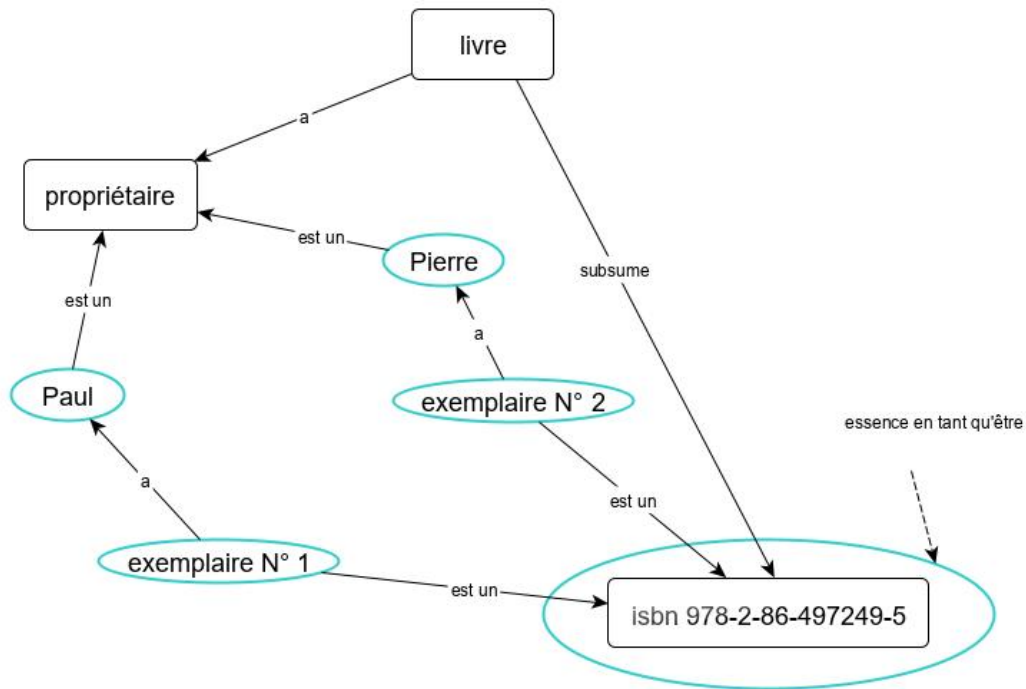


L'expression " ((monde get: #Toupy) getEtreAttribut: #abri) getNom " donne :  
#niche\_0

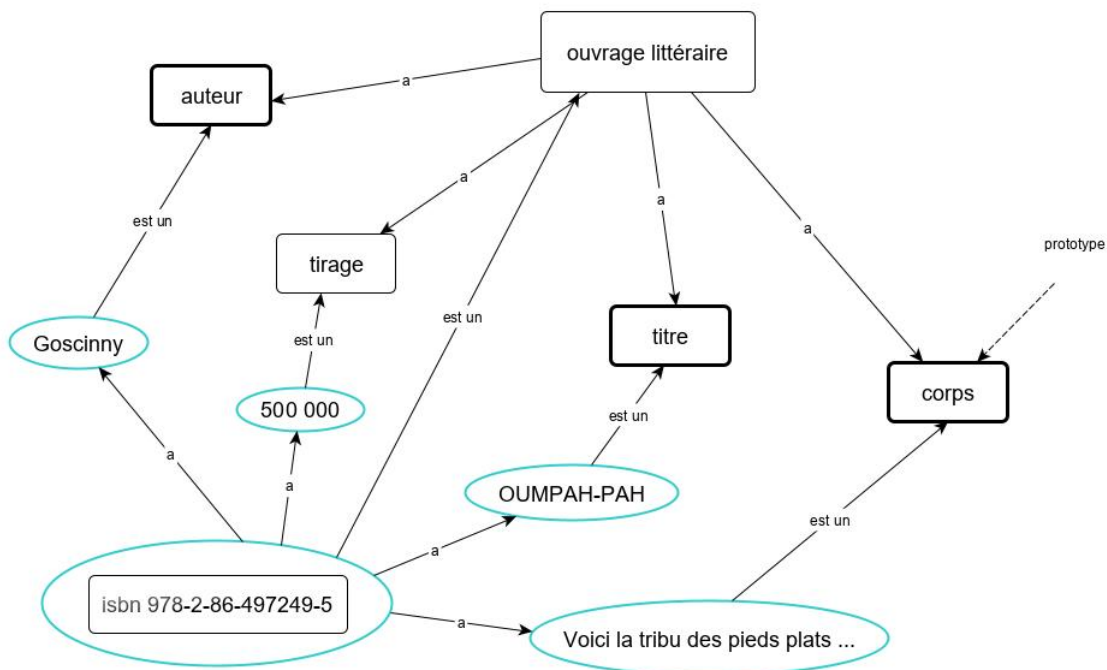


## Exemple 7 : sur la notion de méta essence (essence d'une essence)

Le graphe suivant correspond à la définition de l'essence 'isbn 978-2-86-497249-5' subsumée par l'essence livre, avec deux exemplaires dont les propriétaires respectifs sont Paul et Pierre :



En tant qu'être, l'essence "isbn 978-2-86-497249-5" est instance de l'essence "ouvrage littéraire" (sa méta essence) :



Voici la représentation du contenu de ces deux graphes dans ICEO :

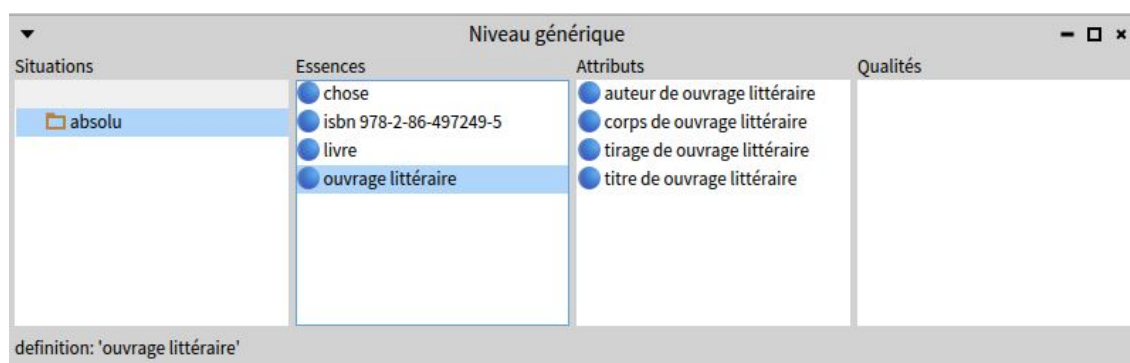
```
iceo definition: 'ouvrage littéraire'.
iceo definitionAttribut: #corps de: (absolu get: 'ouvrage littéraire').
    isPrototype: true.
iceo definitionAttribut: #tirage de: (absolu get: 'ouvrage littéraire').
iceo definitionAttribut: #auteur de: (absolu get: 'ouvrage littéraire').
    isPrototype: true.
iceo definitionAttribut: #titre de: (absolu get: 'ouvrage littéraire').
    isPrototype: true.
iceo definition: #livre.
iceo definitionAttribut: #propriétaire de: livre.
```

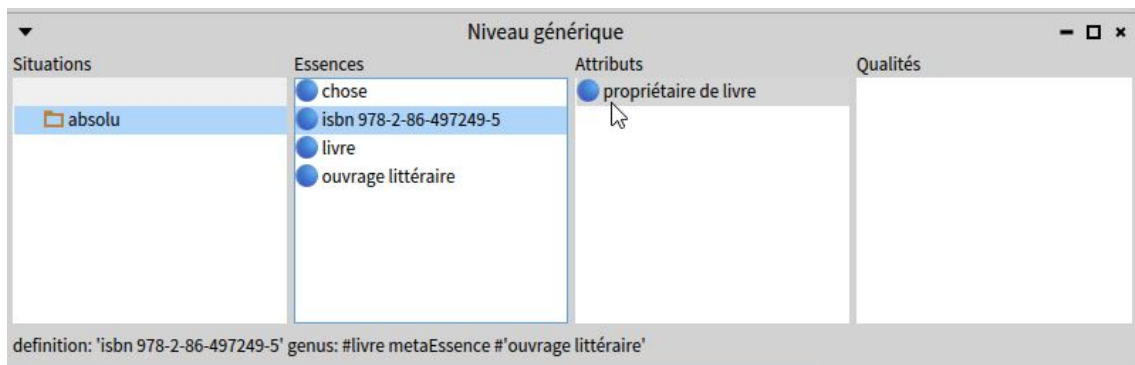
```
iceo definition: 'isbn 978-2-86-497249-5' genus: livre
    metaEssence: (absolu get: 'ouvrage littéraire').
```

```
(absolu get: 'isbn 978-2-86-497249-5') attributionEtre: '500000' .
essence: ((absolu get: 'ouvrage littéraire') getEssenceAttribut: #tirage).
(absolu get: 'isbn 978-2-86-497249-5') attributionEtre: #Goscinny.
essence: ((absolu get: 'ouvrage littéraire') getEssenceAttribut: #auteur).
(absolu get: 'isbn 978-2-86-497249-5') attributionEtre: 'OUMPAH-PAH'.
essence: ((absolu get: 'ouvrage littéraire') getEssenceAttribut: #titre).
(absolu get: 'isbn 978-2-86-497249-5') attributionEtre: 'Voici la tribu des
    pieds plats...'.
essence: ((absolu get: 'ouvrage littéraire') getEssenceAttribut: #corps).
iceo soit: #Paul essence: (livre getEssenceAttribut: #propriétaire).
iceo soit: #Exemplaire_1 essence: (absolu get: 'isbn 978-2-86-497249-5') .
(monde get: #Exemplaire_1) attributionEtre: (monde get: #Paul).
iceo soit: #Pierre essence: (livre getEssenceAttribut: #propriétaire).
iceo soit: #Exemplaire_2 essence: (absolu get: 'isbn 978-2-86-497249-5').
(monde get: #Exemplaire_2) attributionEtre: (monde get: #Pierre).
```

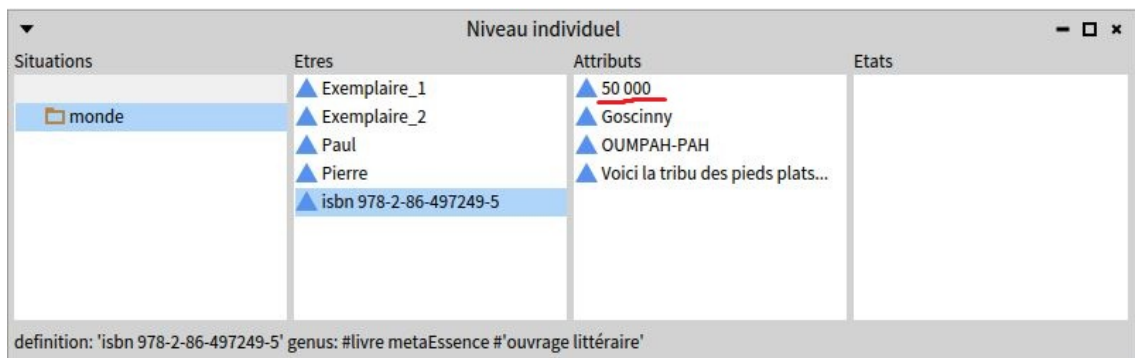
L'instruction encadrée définit l'essence "isbn 978-2-86-497249-5" comme subsumée par livre et instance d'ouvrage littéraire (sa méta essence).

Dans un SgBrowser nous pouvons visualiser les attributs des essences ouvrage littéraire et isbn 978-2-86-497249-5 :

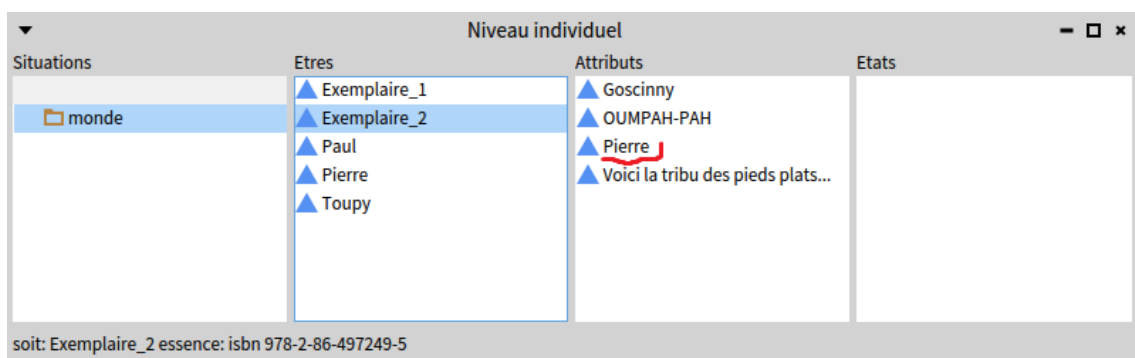
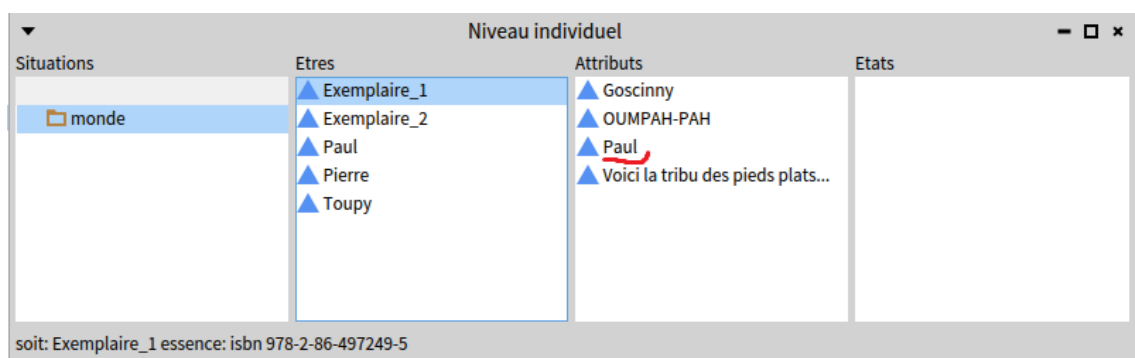




et dans un SiBrowser les attributs de l'essence isbn 978-2-86-497249-5 en tant qu'être :



et les attributs des exemplaire 1 et 2 :



On voit que les deux exemplaires ont les mêmes attributs titre, auteur et corps (qualifiés de prototypes au niveau de ouvrage littéraire) mais que leur attribut propriétaire, que ne possède pas isbn 978-2-86-497249-5, est différent.

Par contre, il n'ont pas l'attribut tirage que possède isbn 978-2-86-497249-5.

Notons q'une petite incorrection apparait dans cet exemple, car il n'est pas vraiment correct de considérer Pierre ou Paul comme attributs d'un exemplaire du livre.

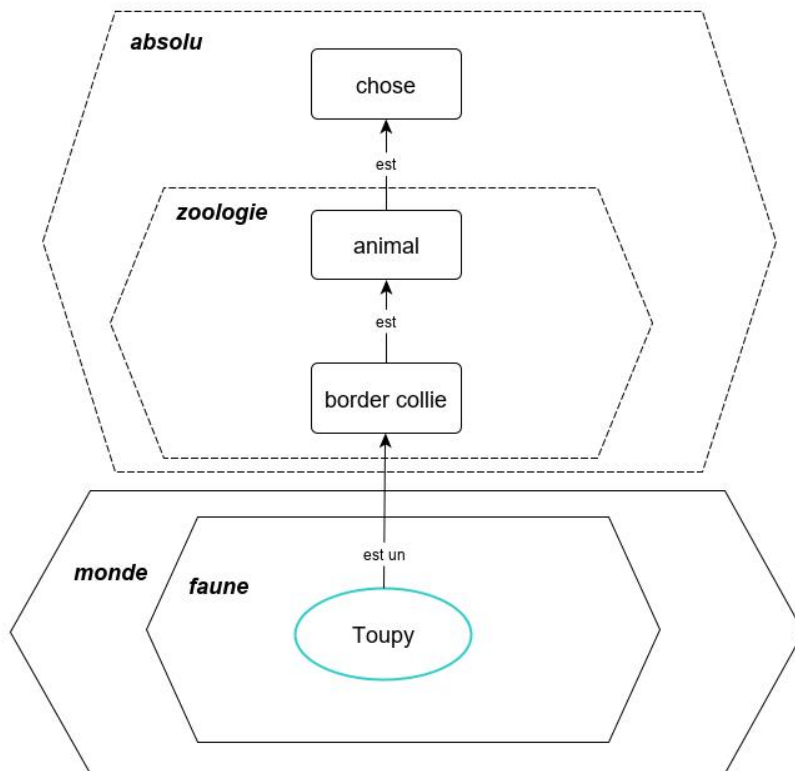
En fait, ce sont Pierre et Paul qui sont chacun propriétaire d'un exemplaire du livre.

Dire qu'un livre a un propriétaire est en fait un raccourci de langage pour exprimer qu'un exemplaire du livre appartient à une personne qui en est propriétaire.

Nous verrons comment corriger ce point dans l'exemple 14.

## Exemple 8 : sur les notions de situation générique et de situation individuelle

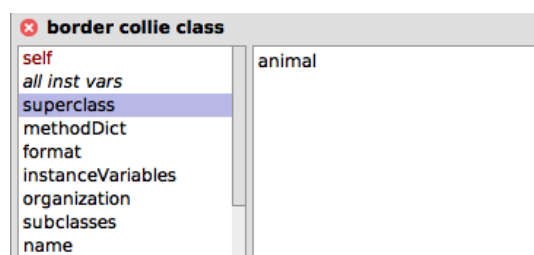
Considérons le graphe suivant :



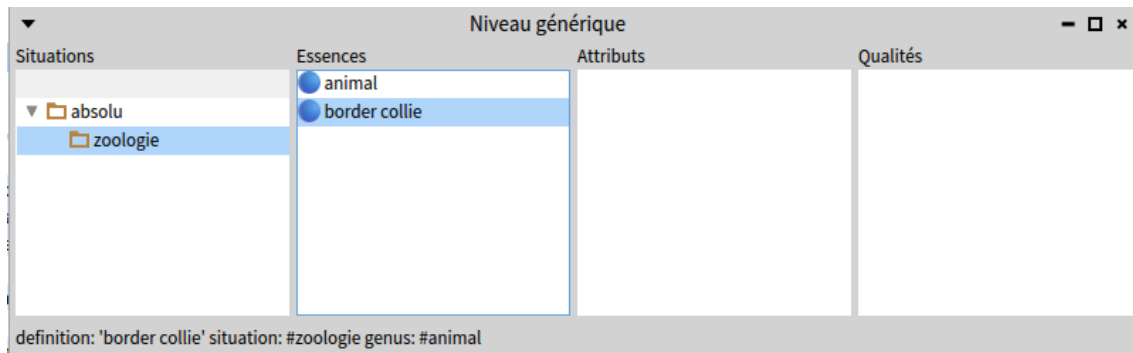
```
iceo definitionSituation: #zoologie.
iceo definition: #animal situation: zoologie.
iceo definition: 'border collie' situation: zoologie genus: (zoologie get:
    #animal).
iceo soit: #faune situationGenerique: zoologie.
iceo soit: #Toupy essence: (zoologie get: 'border collie ')
    situationIndividuelle: (monde getSituation: #faune).
```

L'essence nommée "border collie" est subsumée par l'essence animal dans une situation générique nommée "zoologie", et un être "Toupy" est créé dans une situation individuelle nommée "faune". La situation générique zoologie est incluse dans l'absolu et la situation individuelle faune, instance de la situation zoologie, est incluse dans le monde.

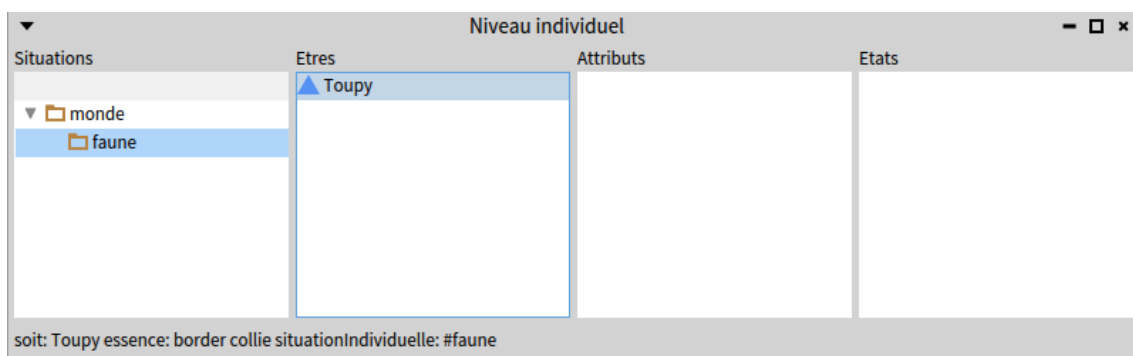
L'inspection de "border collie" en utilisant l'expression "zoologie get: 'border collie' " permet de vérifier qu'elle est bien subsumée par l'essence animal :



et sa situation générique est zoologie :

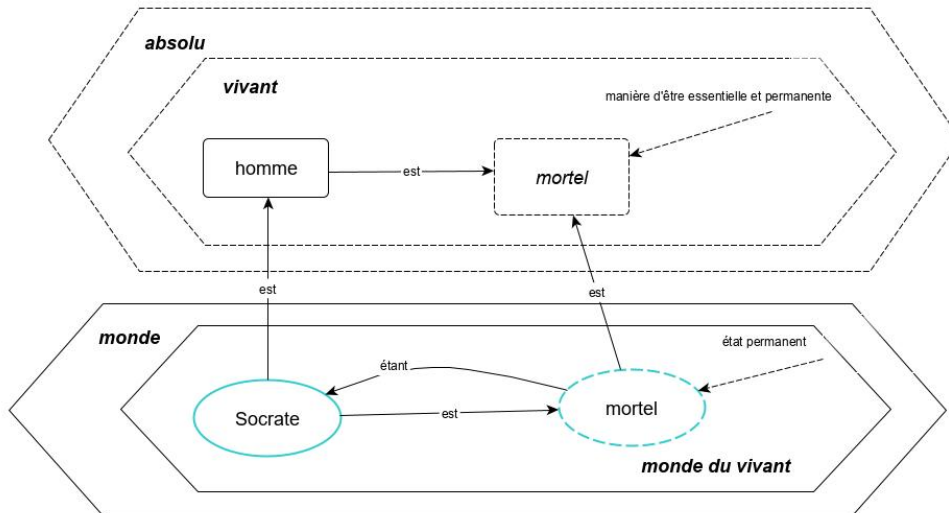


L'expression " (monde getSituation: #faune) get: #Toupy " permet de vérifier que Toupy est bien une instance de border collie présente dans la situation individuelle faune, ce qui se vérifie dans la fenêtre suivante :



## Exemple 9 : sur la notion de manière d'être essentielle et permanente

Considérons le graphe suivant :

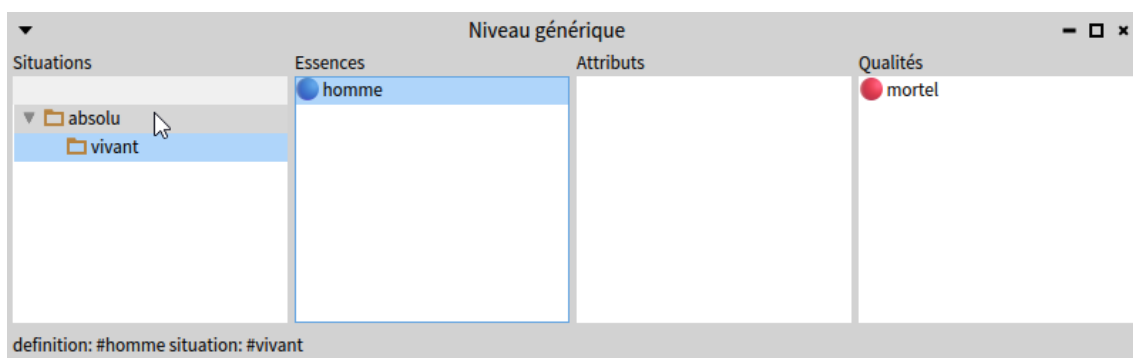


```

iceo definitionSituation: #vivant.
iceo definition: #homme situation: vivant.
iceo definitionQualiteEssentielle: #mortel pour: (vivant get: #homme)
    effectivite: #permanente.
iceo soit: 'monde du vivant' situationGenerique: vivant.
iceo soit: #Socrate essence: (vivant get: #homme) situationIndividuelle: (
    monde get: 'monde du vivant').
    
```


La troisième instruction définit la manière d'être "mortel" comme essentielle et permanente pour homme.





C'est ce qui se vérifie dans la fenêtre suivante :



La petite icône rouge associée à la qualité "mort" indique qu'il s'agit d'une qualité essentielle.

La fenêtre suivante montre que Socrate, en tant qu'homme, est bien mortel :

▼  Niveau individuel

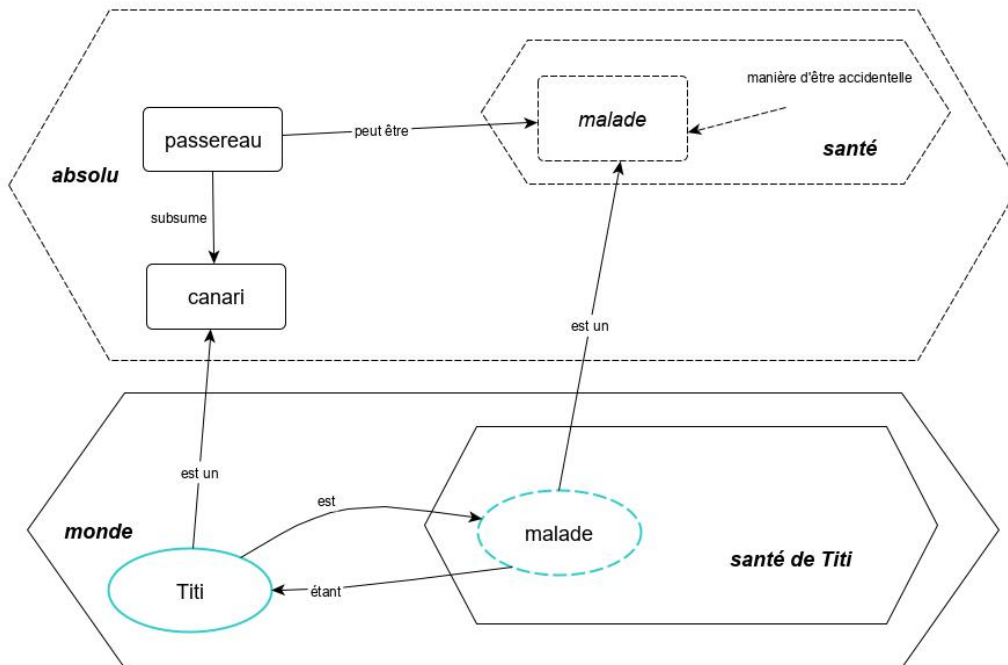
Situations	Etres	Attributs	Etats
▼  monde  monde du vivant	 Socrate		 mortel

soit: Socrate essence: homme situationIndividuelle: #'monde du vivant'



## Exemple 10 : sur la notion de manière d'être accidentelle

Considérons le graphe suivant :

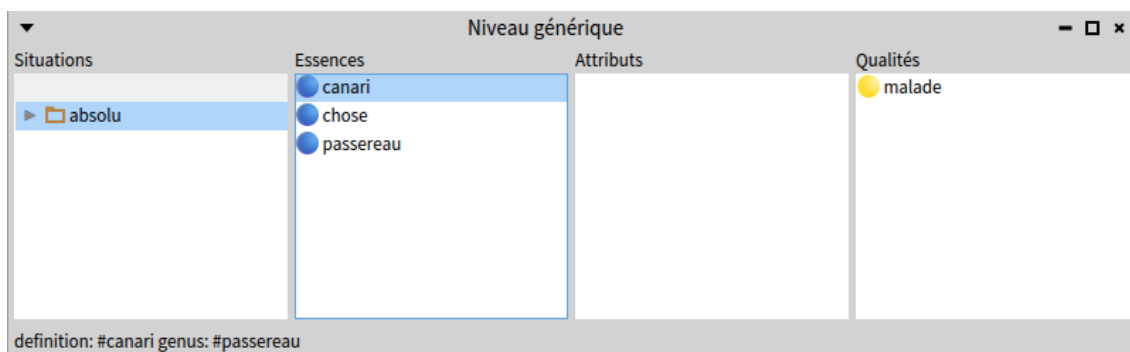


```

iceo definition: #passereau.
iceo definition: #canari genus: passereau.
iceo definitionSituation: #santé.
iceo definitionQualite: #malade situation: santé. passereau peutEtre:
    (santé get: #malade).
iceo soit: #Titi essence: canari.
iceo soit: 'santé de Titi' situationGenerique: santé.
(monde get: #Titi) affecteEtat: (santé get: #malade) dansSituation: (monde
    get: 'santé de Titi').
    
```

L'essence passereau peut avoir la manière d'être accidentelle "malade" (statut par défaut de la manière d'être "malade" définie à la quatrième instruction).

Cette manière d'être est héritée par l'essence canari qui est subsumée par passereau.



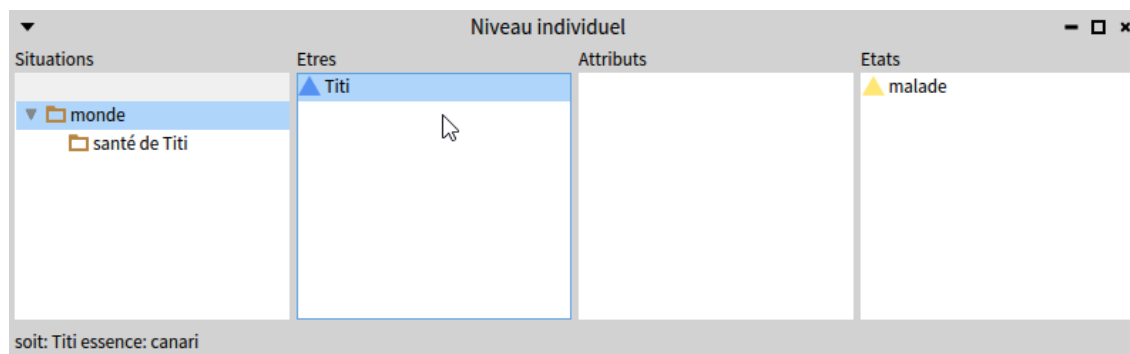
La petite icône jaune associée à la qualité "malade" indique qu'il s'agit d'une qualité accidentelle.

Pour que Titi soit malade, il faut que cet état lui soit explicitement affecté, car il s'agit d'une manière d'être accidentelle.

C'est ce qui est fait dans la dernière instruction.

L'état malade de Titi est instance de la manière d'être malade.

C'est ce que confirme la fenêtre suivante :

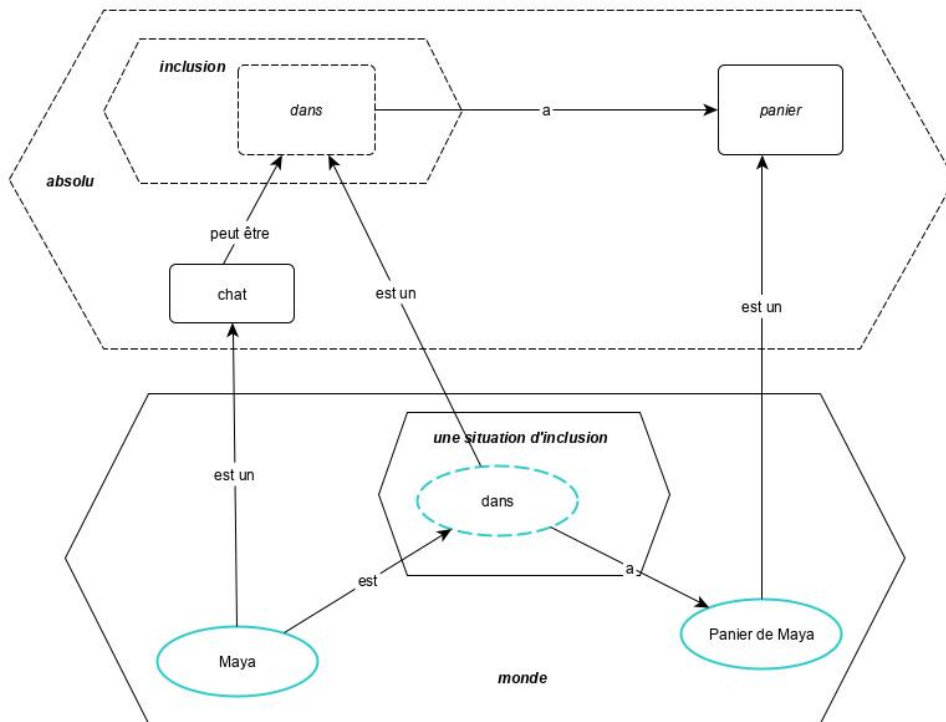


Le petit triangle jaune associé à l'état "malade" de Titi indique qu'il s'agit d'un état accidentel.

Souhaitons à Titi un prompt rétablissement.

## Exemple 11 : sur les attributs d'une manière d'être

Considérons le graphe suivant:

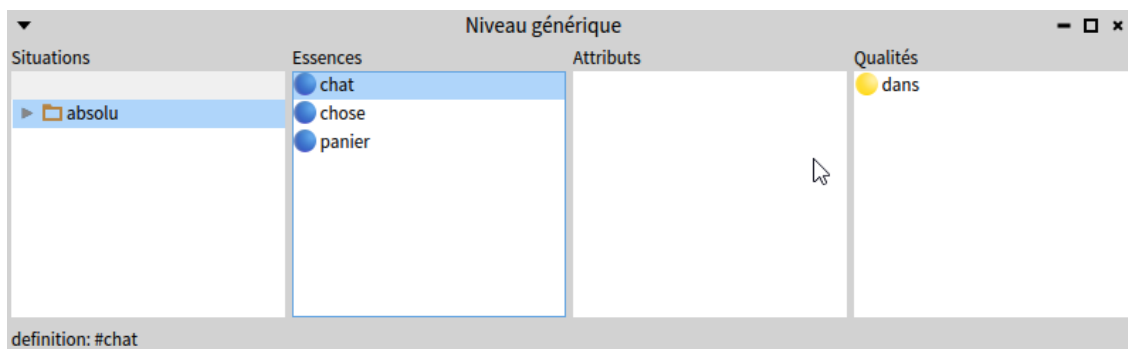


Il est la représentation de la phrase : "La chatte Maya est dans son panier"

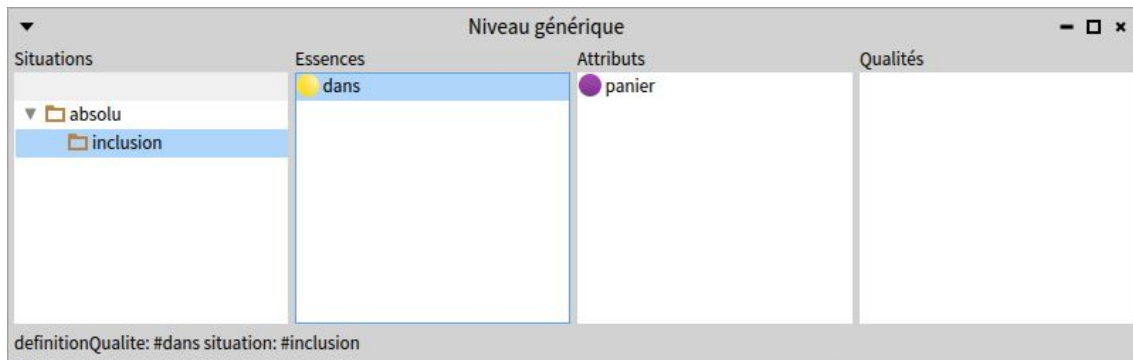
```

iceo definition: #chat.
iceo definition: #panier.
iceo definitionSituation: #inclusion.
iceo definitionQualite: #dans situation: inclusion.
(inclusion get: #dans) referenceEssence: panier.
chat peutEtre: (inclusion get: #dans).
iceo soit: 'une situation d''inclusion' situationGenerique: inclusion.
iceo soit: #Maya essence: chat.
iceo soit: 'Panier de Maya' essence: panier.
(monde get: #Maya) affecteEtat: (inclusion get: #dans) dansSituation: (
    monde get: 'une situation d''inclusion').
((monde get: #Maya) getEtat: #dans) attributionEtre: (monde get: 'Panier de
    Maya' ).
  
```

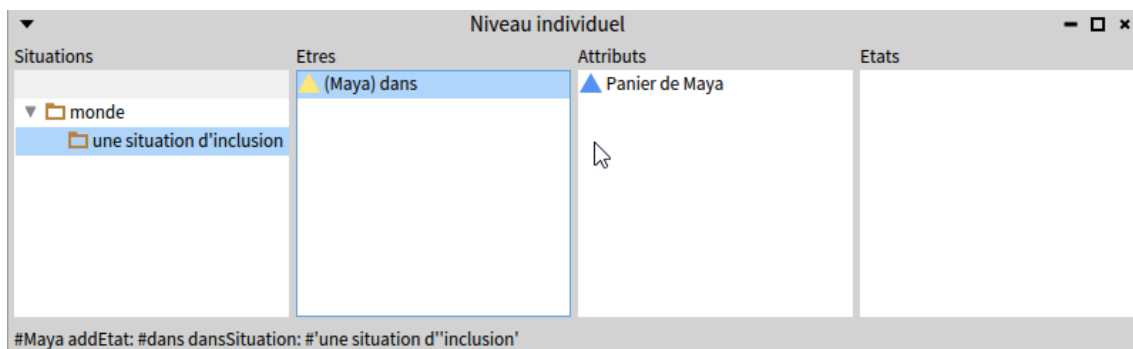
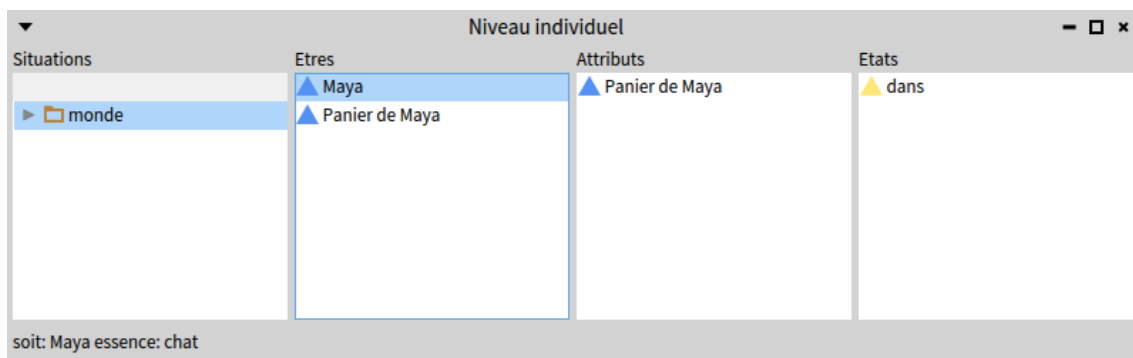
La fenêtre suivante montre que "dans" est une qualité accidentelle de chat :



et celle-ci que l'essence panier est attribut de la manière d'être "dans" :



Celles-ci montrent que Maya est "dans" son panier :



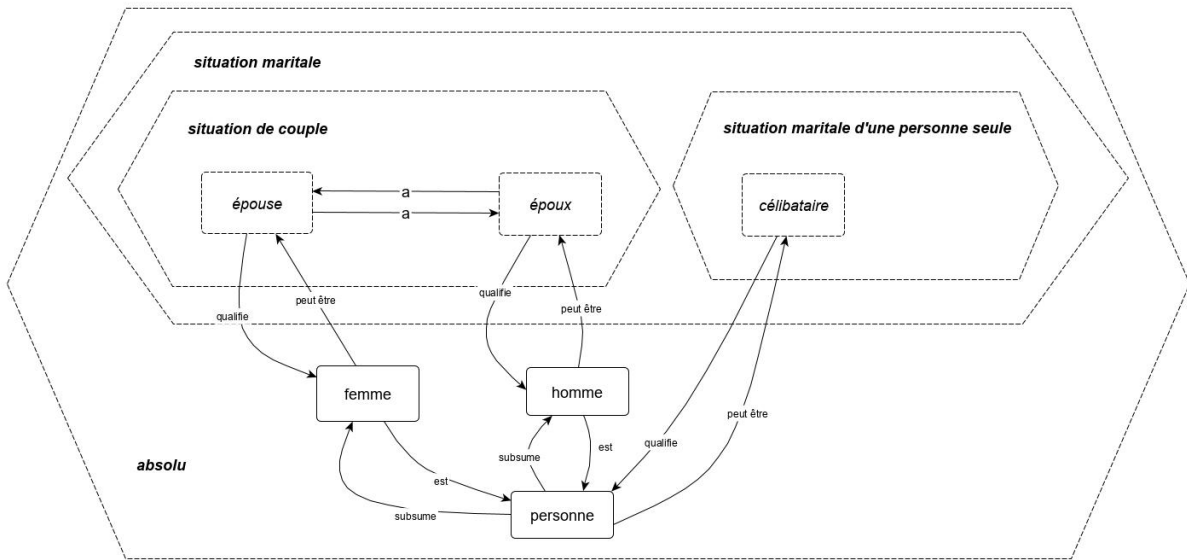
L'état "dans" qui s'affiche en tant qu'être (ce qui est normal, car un état est un être !) est précédé du nom de son étant (l'être qui est dans cet état) placé entre parenthèses.

L'expression "`(monde get: #Maya) getEtresAttributsEnTantQue: (inclusion get: #dans)`" donne : an OrderedCollection(panier de Maya)

Suivant la même logique, nous pourrions représenter toutes les prépositions de la langue française (sur, avec, entre, ...)

## Exemple 12 : sur les relations entre manières d'être

Considérons le graphe suivant :



```
iceo definition: #personne.
iceo definition: #femme genus: personne.
iceo definition: #homme genus: personne.
iceo definition: #maire genus: personne.
iceo definitionSituation: 'situation de couple'.
iceo definitionSituation: 'situation de personne seule'.
iceo definitionQualite: #épouse situation: (absolu getSituation: 'situation
de couple').
iceo definitionQualite: #époux situation: (absolu getSituation: 'situation
de couple').
iceo definitionQualite: #célibataire situation: (absolu getSituation: '
situation de personne seule').
```

```
((absolu getSituation: 'situation de couple') get: #épouse)
  associationQualite: ((absolu getSituation: 'situation de couple')
    get: #époux).
```

```
femme peutEtre: ((absolu getSituation: 'situation de couple') get: #épouse).
femme peutEtre: ((absolu getSituation: 'situation de personne seule') get:
  #célibataire).
homme peutEtre: ((absolu getSituation: 'situation de couple') get: #époux).
homme peutEtre: ((absolu getSituation: 'situation de personne seule') get:
  #célibataire).
```

L'instruction encadrée définit une relation bidirectionnelle entre les manières d'être épouse et époux.

Ceci peut être vérifié dans les fenêtres suivantes :

Niveau générique			
Situations	Essences	Attributs	Qualités
<ul style="list-style-type: none"> <li>absolu               <ul style="list-style-type: none"> <li>situation de couple</li> <li>situation de personne seule</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>épouse</li> <li>époux</li> </ul>	<ul style="list-style-type: none"> <li>époux</li> </ul>	

definitionQualite: #épouse situation: #'situation de couple'

Niveau générique			
Situations	Essences	Attributs	Qualités
<ul style="list-style-type: none"> <li>absolu               <ul style="list-style-type: none"> <li>situation de couple</li> <li>situation de personne seule</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>épouse</li> <li>époux</li> </ul>	<ul style="list-style-type: none"> <li>épouse</li> </ul>	

definitionQualite: #époux situation: #'situation de couple'

Les manières d'être possibles pour homme et femme apparaissent dans les fenêtres suivantes :

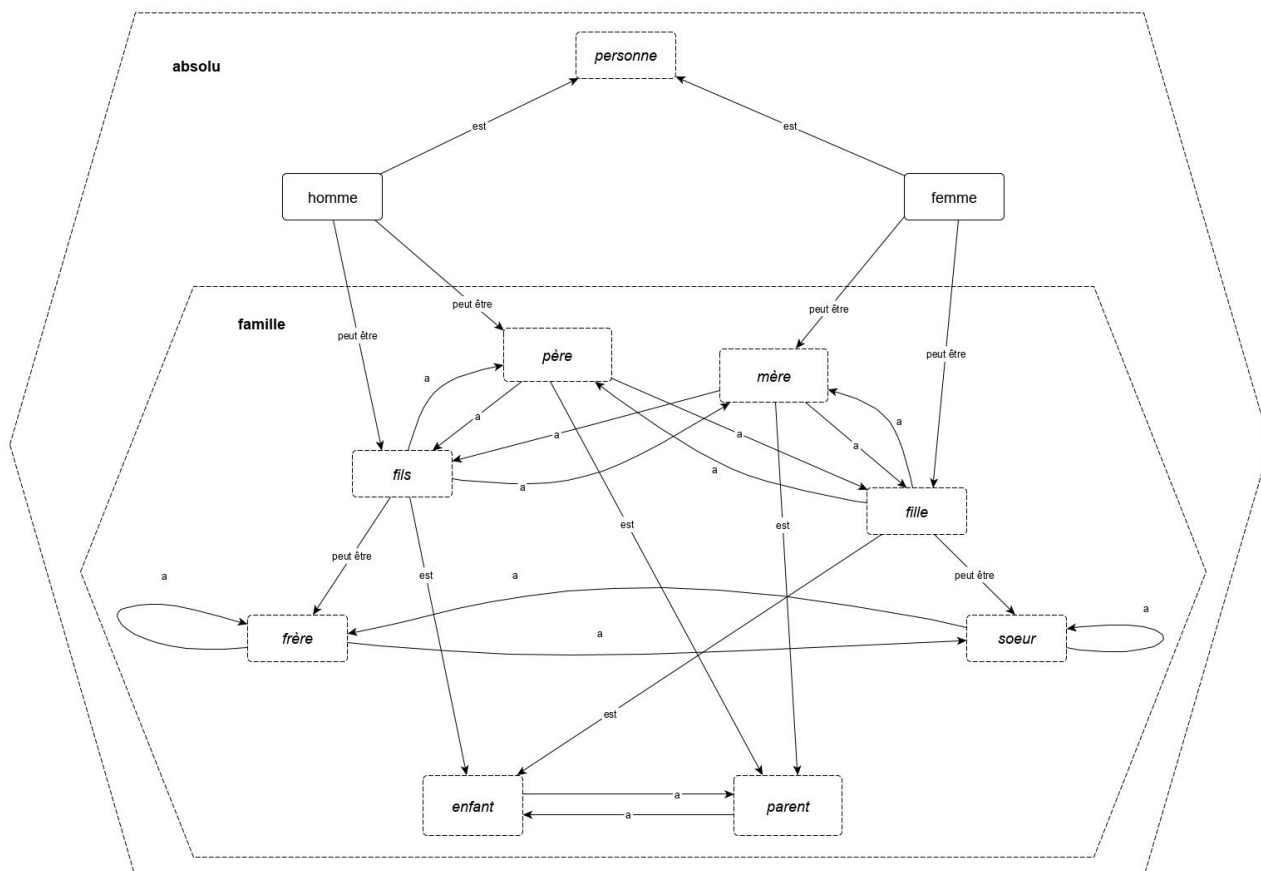
Niveau générique			
Situations	Essences	Attributs	Qualités
<ul style="list-style-type: none"> <li>absolu               <ul style="list-style-type: none"> <li>situation de couple</li> <li>situation de personne seule</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>chose</li> <li>femme</li> <li>homme</li> <li>personne</li> </ul>		<ul style="list-style-type: none"> <li>célibataire</li> <li>épouse</li> </ul>

definition: #femme genus: #personne

Niveau générique			
Situations	Essences	Attributs	Qualités
<ul style="list-style-type: none"> <li>absolu               <ul style="list-style-type: none"> <li>situation de couple</li> <li>situation de personne seule</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>chose</li> <li>femme</li> <li>homme</li> <li>personne</li> </ul>		<ul style="list-style-type: none"> <li>célibataire</li> <li>époux</li> </ul>

definition: #homme genus: #personne

### Exemple 13 : sur la subsumption des manières d'être



```

iceo definition: #personne.
iceo definition: #femme genus: personne.
iceo definition: #homme genus: personne.
iceo definitionSituation: 'famille '.
iceo definitionQualite: #parent situation: (absolu getSituation: #famille).
iceo definitionQualite: #enfant situation: (absolu getSituation: #famille).
iceo definitionQualite: #soeur situation: (absolu getSituation: #famille).
iceo definitionQualite: #frère situation: (absolu getSituation: #famille).
iceo definitionQualite: #fils situation: (absolu getSituation: #famille)
    genus: ((absolu getSituation: #famille) get: #enfant).
iceo definitionQualite: #père situation: (absolu getSituation: #famille)
    genus: ((absolu getSituation: #famille) get: #parent).
iceo definitionQualite: #fille situation: (absolu getSituation: #famille)
    genus: ((absolu getSituation: #famille) get: #enfant).
iceo definitionQualite: #mère situation: (absolu getSituation: #famille)
    genus: ((absolu getSituation: #famille) get: #parent).
((absolu getSituation: #famille) get: #parent) associationQualite: ((absolu
    getSituation: #famille) get: #enfant).
((absolu getSituation: #famille) get: #père) associationQualite: ((absolu
    getSituation: #famille) get: #fils).
((absolu getSituation: #famille) get: #père) associationQualite: ((absolu
    getSituation: #famille) get: #fille).
((absolu getSituation: #famille) get: #mère) associationQualite: ((absolu
    getSituation: #famille) get: #fils).

```

```

((absolu getSituation: #famille) get: #mère) associationQualite: ((absolu
    getSituation: #famille) get: #fille).
((absolu getSituation: #famille) get: #frère) associationQualite: ((absolu
    getSituation: #famille) get: #soeur).
((absolu getSituation: #famille) get: #frère) associationQualite: ((absolu
    getSituation: #famille) get: #frère).
((absolu getSituation: #famille) get: #soeur) associationQualite: ((absolu
    getSituation: #famille) get: #soeur).
homme peutEtre: ((absolu getSituation: #famille) get: #père).
homme peutEtre: ((absolu getSituation: #famille) get: #fils).
femme peutEtre: ((absolu getSituation: #famille) get: #mère).
femme peutEtre: ((absolu getSituation: #famille) get: #fille).
((absolu getSituation: #famille) get: #fille) peutEtre: ((absolu
    getSituation: #famille) get: #soeur).
((absolu getSituation: #famille) get: #fils) peutEtre: ((absolu
    getSituation: #famille) get: #frère).

```

Dans cet exemple, certaines manières d'être en subsument d'autres.

Ainsi " ((absolu getSituation: #famille) get: #fils) getGenus " donne : enfant

Notons également qu'une manière d'être peut adopter d'autres manières d'être.

Ainsi " ((absolu getSituation: #famille) get: #fils) getQualites " donne : an OrderedCollection(frère)

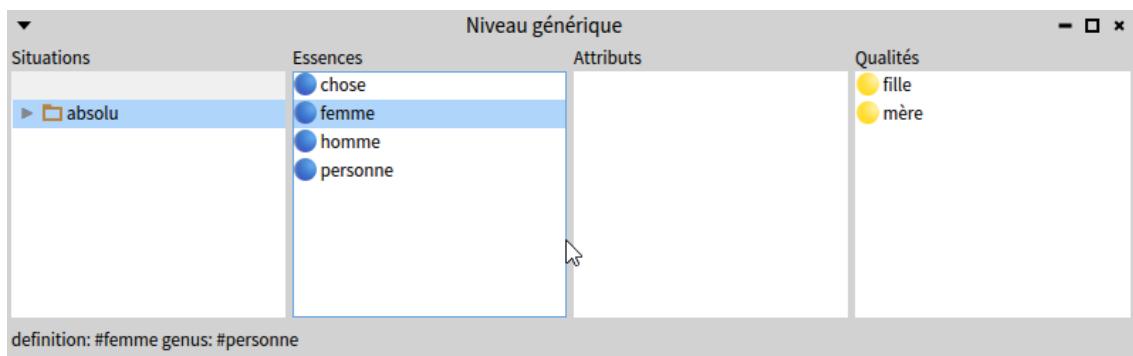
Et, comme nous l'avons déjà vu, une manière d'être peut avoir pour attributs d'autres manières d'être.

Ainsi, " ((absolu getSituation: #famille) get: #frère) getDifferentia " donne : an OrderedCollection(soeur frère)

Cet exemple illustre le fait qu'une manière d'être peut être attribut d'elle même.

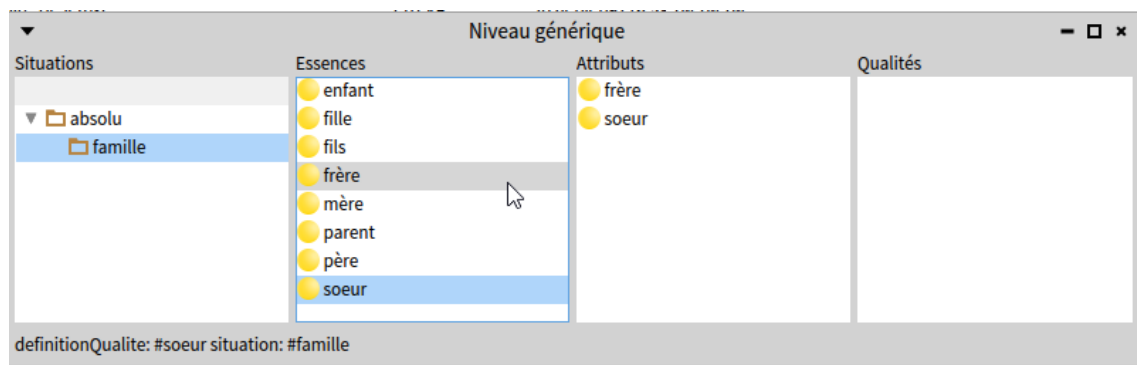
(Rappelons qu'une essence qui n'est pas une manière d'être ne peut être attribut d'elle même).

Un browser ouvert au niveau générique permet de vérifier par exemple les manières d'être possibles d'une femme :



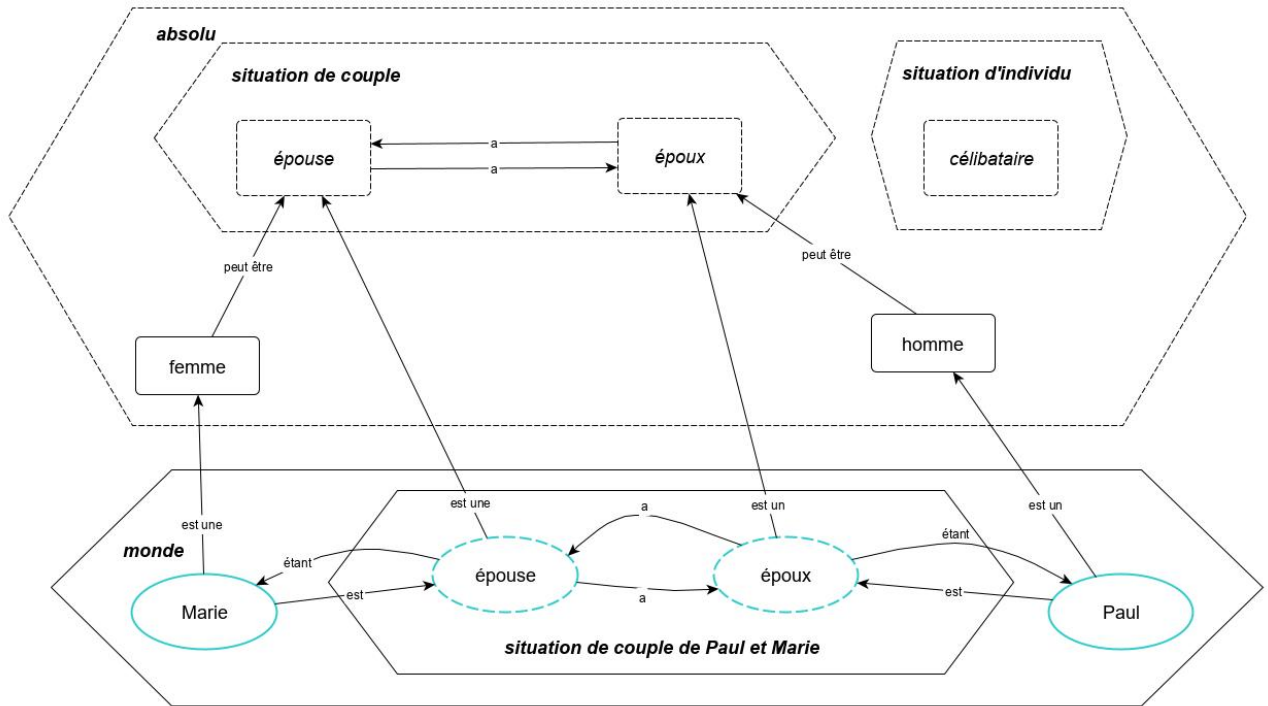
ou les attributs possibles de la manière d'être frère :





## Exemple 14 : sur la relation entre états

Considérons le graphe suivant :



```

iceo definition: #femme.
iceo definition: #homme.
iceo definitionSituation: 'situation de couple'
iceo definitionQualite: #épouse situation: (absolu getSituation: 'situation
de couple').
iceo definitionQualite: #époux situation: (absolu getSituation:
'situation de couple').
((absolu getSituation: 'situation de couple') get: #épouse)
  associationQualite: ((absolu getSituation: 'situation de couple')
    get: #époux).
femme peutEtre: ((absolu getSituation: 'situation de couple')
  get: #épouse).
homme peutEtre: ((absolu getSituation: 'situation de couple')
  get: #époux).
iceo soit: #Marie essence: femme.
iceo soit: #Paul essence: homme.
iceo soit: 'situation de couple de Paul et Marie' situationGenerique: (
  absolu getSituation: 'situation de couple').
(monde get: #Marie) affecteEtat: ((absolu getSituation: 'situation de
couple') get: #épouse) dansSituation: (monde getSituation: 'situation de
couple de Paul et Marie').
(monde get: #Paul) affecteEtat: ((absolu getSituation: 'situation de
couple') get: #époux) dansSituation: (monde getSituation: 'situation de
couple de Paul et Marie').

(((monde get: #Paul) getEtat: #époux dansSituation: (monde getSituation:
'situation de couple de Paul et Marie')) associationEtat: ((monde
get: #Marie) getEtat: #épouse dansSituation: (monde getSituation: '
situation de couple de Paul et Marie')))).
  
```

L'instruction encadrée définit une relation bidirectionnelle entre les états épouse et époux

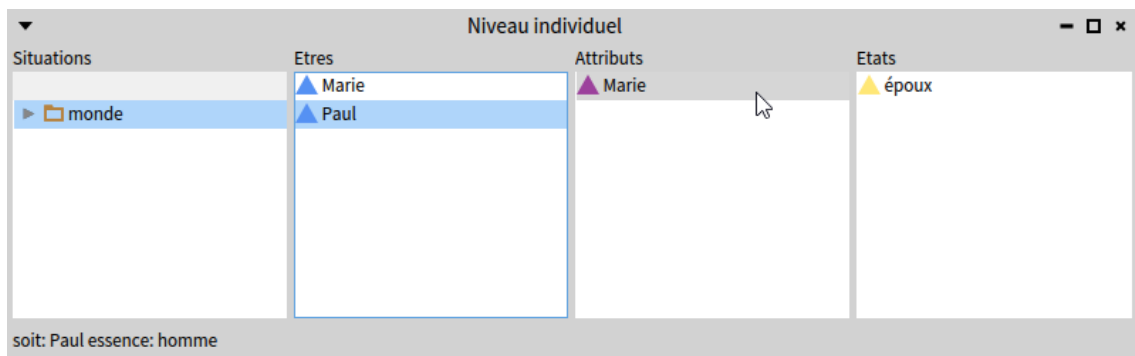
qui est rendue possible par la relation bidirectionnelle entre les manières d'être épouse et époux au niveau générique.

Dans cet exemple, Marie et Paul ne sont pas attributs l'un de l'autre, mais l'expression

```
" (monde get: #Paul) getEtresAttributsEnTantQue: ((absolu getSituation:
'situation de couple') get: #époux) "
```

donne pourtant : an OrderedCollection(Marie).

Cet attribut lui est conféré par son état d'époux de Marie (attribut qu'il perdrait en cas de divorce ...)



La petite icône violette devant l'attribut Marie indique qu'il ne s'agit pas d'un attribut propre, mais acquis par Paul en sa qualité d'époux.

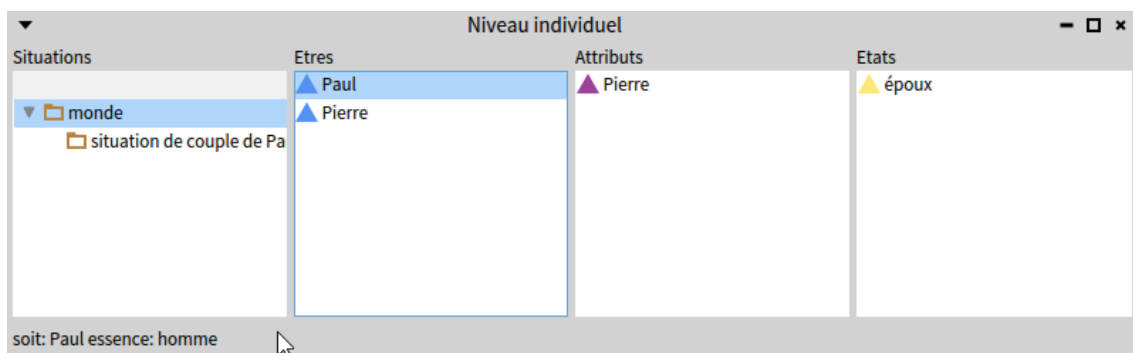
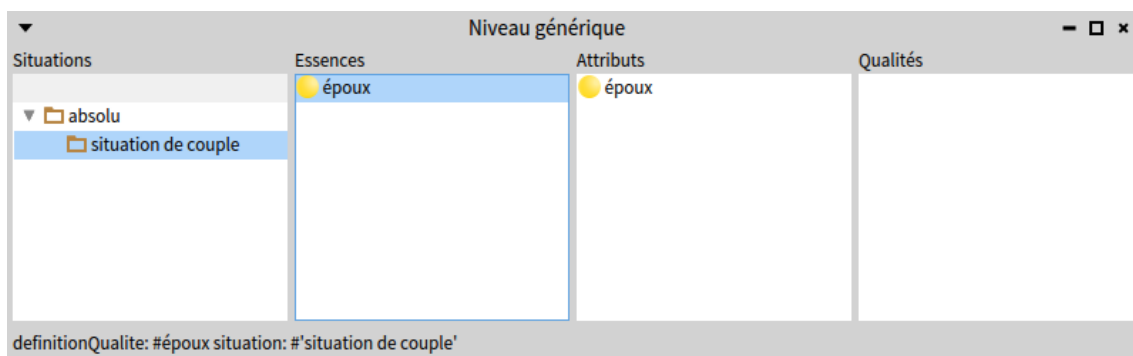
## Exemple 14\_2

Cet exemple est une variante du précédent qui représente un couple homosexuel :

```

iceo definition: #homme.
iceo definitionSituation: 'situation de couple'.
iceo definitionQualite: #époux situation: (absolu getSituation: 'situation
de couple').
((absolu getSituation: 'situation de couple') get: #époux)
associationQualite: ((absolu getSituation: 'situation de couple')
get: #époux).
homme peutEtre: ((absolu getSituation: 'situation de couple') get: #époux).
iceo soit: #Pierre essence: homme.iceo soit: #Paul essence: homme.
iceo soit: 'situation de couple de Paul et Pierre' situationGenerique:
(absolu getSituation: 'situation de couple').
(monde get: #Pierre) affecteEtat: ((absolu getSituation: 'situation de
couple') get: #époux) dansSituation: (monde getSituation: 'situation de
couple de Paul et Pierre').
(monde get: #Paul) affecteEtat: ((absolu getSituation: 'situation de
couple') get: #époux) dansSituation: (monde getSituation: 'situation de
couple de Paul et Pierre').
(((monde get: #Paul) getEtat: #époux dansSituation: (monde getSituation: '
situation de couple de Paul et Pierre')) associationEtat: ((monde get:
#Pierre) getEtat: #époux dansSituation: (monde getSituation: 'situation
de couple de Paul et Pierre')))).

```

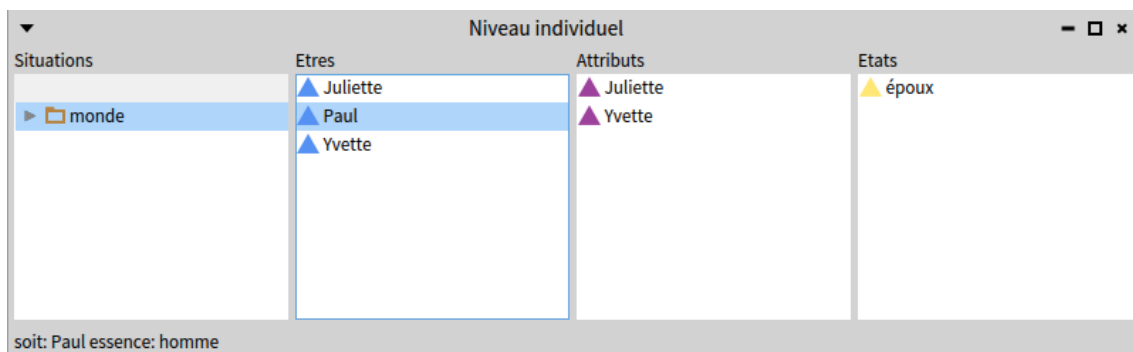


## Exemple 14\_3

Cet exemple représente le cas d'un homme polygame :

```
iceo definition: #femme.
iceo definition: #homme.
iceo definitionSituation: 'situation de polycouple '.
iceo definitionQualite: #épouse situation: (absolu getSituation: 'situation
  de polycouple ').
iceo definitionQualite: #époux situation: (absolu getSituation: 'situation
  de polycouple ').
((absolu getSituation: 'situation de polycouple ') get: #épouse)
  associationQualite: ((absolu getSituation: 'situation de polycouple ')
    get: #époux).
femme peutEtre: ((absolugetSituation: 'situation de polycouple ')
  get: #épouse).
homme peutEtre: ((absolu getSituation: 'situation de polycouple ')
  get: #époux).
iceo soit: #Yvette essence: femme.
iceo soit: #Juliette essence: femme.
iceo soit: #Paul essence: homme.
iceo soit: 'situation de polycouple de Paul' situationGenerique: (absolu
  getSituation: 'situation de polycouple ').
(monde get: #Yvette) affecteEtat: ((absolu getSituation: 'situation de
  polycouple ') get: #épouse) dansSituation: (monde getSituation: '
  situation de polycouple de Paul').
(monde get: #Juliette) affecteEtat: ((absolu getSituation: 'situation de
  polycouple ') get: #épouse) dansSituation: (monde getSituation: '
  situation de polycouple de Paul').
(monde get: #Paul) affecteEtat: ((absolu getSituation: 'situation de
  polycouple ') get: #époux) dansSituation: (monde getSituation: 'situation
  de polycouple de Paul').
```

```
((monde get: #Paul) getEtat: #époux dansSituation: (monde getSituation:
  'situation de polycouple de Paul')) associationEtat: ((monde get: #
  Yvette) getEtat: #épouse dansSituation: (monde getSituation: '
  situation de polycouple de Paul'))).
(((monde get: #Paul) getEtat: #époux dansSituation: (monde getSituation:
  'situation de polycouple de Paul')) associationEtat: ((monde get: #
  Juliette) getEtat: #épouse dansSituation: (monde getSituation: '
  situation de polycouple de Paul'))).
```



On constate que Paul en tant qu'époux "possède" deux épouses.

## Exemple 14\_4

Cet exemple est une variante de l'exemple précédent où l'état d'époux de Paul est différent dans sa relation par rapport à Juliette ou Yvette :

```

iceo definition: #femme.
iceo definition: #homme.
iceo definitionSituation: 'situation de polycouple '.
iceo definitionQualite: #épouse situation: (absolu getSituation: 'situation
de polycouple ').
iceo definitionQualite: #époux situation: (absolu getSituation: 'situation
de polycouple ').
((absolu getSituation: 'situation de polycouple ') get: #épouse)
associationQualite: ((absolu getSituation: 'situation de polycouple ')
get: #époux).
femme peutEtre: ((absolu getSituation: 'situation de polycouple ')
get: #épouse).
homme peutEtre: ((absolu getSituation: 'situation de polycouple ')
get: #époux).
iceo soit: #Yvette essence: femme.
iceo soit: #Juliette essence: femme.
iceo soit: #Paul essence: homme.
iceo soit: 'situation de polycouple de Paul' situationGenerique: (absolu
getSituation: 'situation de polycouple ').
(monde get: #Yvette) affecteEtat: ((absolu getSituation: 'situation de
polycouple ') get: #épouse) dansSituation: (monde getSituation: '
situation de polycouple de Paul').
(monde get: #Juliette) affecteEtat: ((absolu getSituation: 'situation de
polycouple ') get: #épouse) dansSituation: (monde getSituation: '
situation de polycouple de Paul').

```

```

(monde get: #Paul) affecteEtat: ((absolu getSituation: 'situation de
polycouple ') get: #époux) dansSituation: (monde getSituation: '
situation de polycouple de Paul').
(monde get: #Paul) affecteEtat: ((absolu getSituation: 'situation de
polycouple ') get: #époux) dansSituation: (monde getSituation: '
situation de polycouple de Paul').
(((monde get: #Paul) getEtats: #époux dansSituation: (monde
getSituation: 'situation de polycouple de Paul')) at: 1)
associationEtat: ((monde get: #Yvette) getEtat: #épouse
dansSituation: (monde getSituation: 'situation de polycouple de Paul
')))).
(((monde get: #Paul) getEtats: #époux dansSituation: (monde
getSituation: 'situation de polycouple de Paul')) at: 2)
associationEtat: ((monde get: #Juliette) getEtat: #épouse
dansSituation: (monde getSituation: 'situation de polycouple de Paul
')))).

```

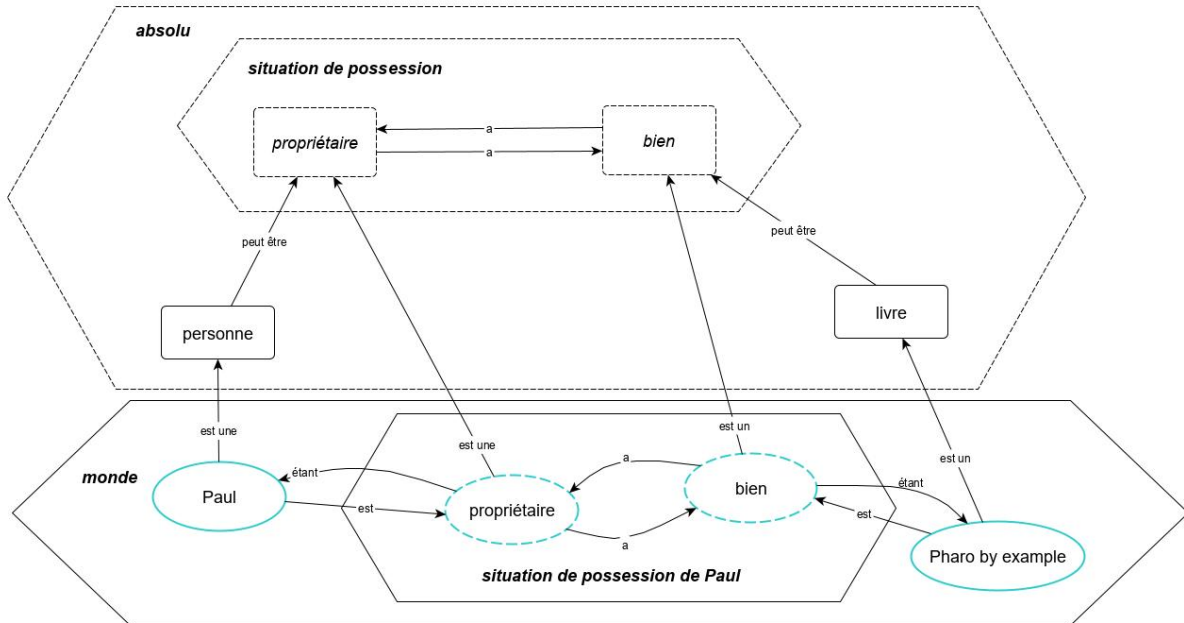
Niveau individuel			
Situations	Etres	Attributs	Etats
monde	<ul style="list-style-type: none"> <li>Juliette</li> <li>Paul</li> <li>Yvette</li> </ul>	<ul style="list-style-type: none"> <li>Juliette</li> <li>Yvette</li> </ul>	<ul style="list-style-type: none"> <li>époux</li> <li>époux</li> </ul>
soit: Paul essence: homme			

Cet exemple est plus informatif que le précédent car, étant donné qu'un état peut être dans un état, il permettrait de préciser que Paul est un mauvais époux pour Yvette et un bon époux pour Juliette !

## Exemple 14\_5

Cet exemple illustre la relation possible entre deux êtres qui ne soit pas une relation d'inclusion (composition).

Il représente le fait qu'une personne puisse être propriétaire d'un livre :



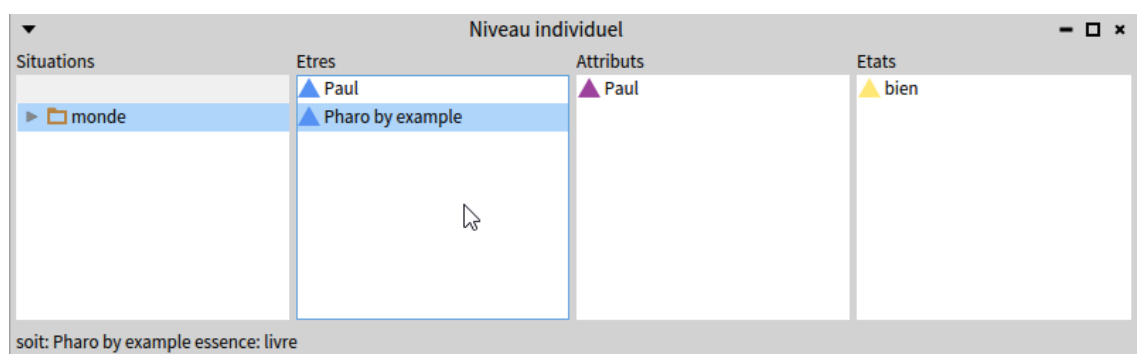
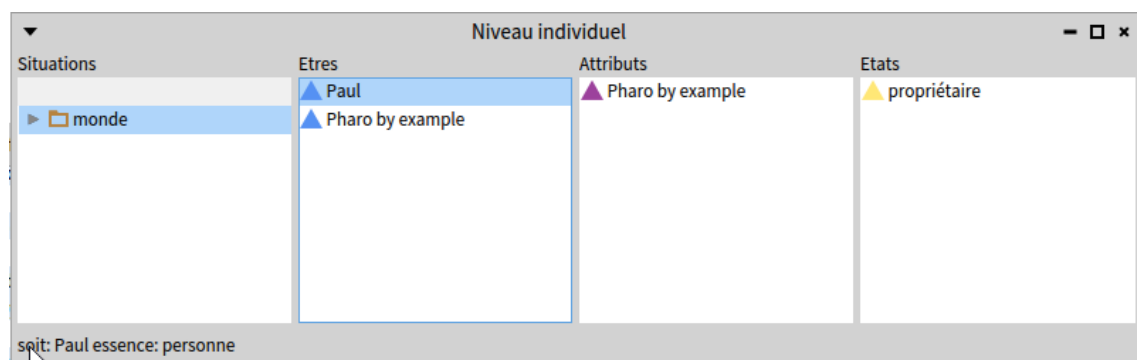
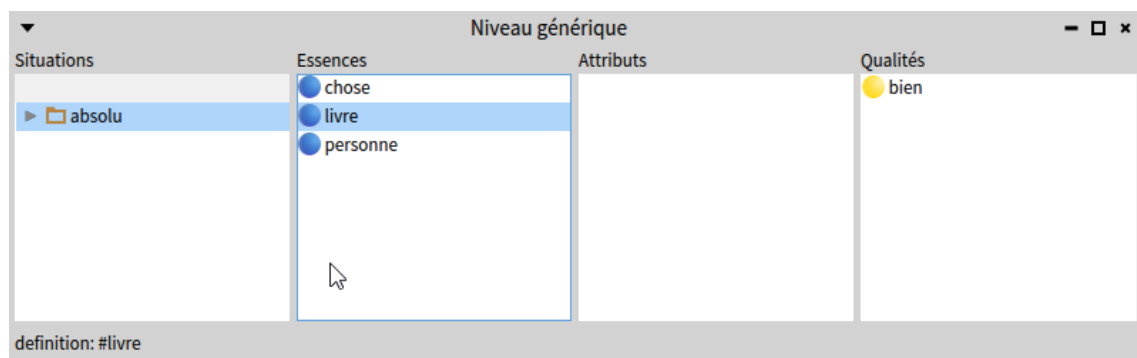
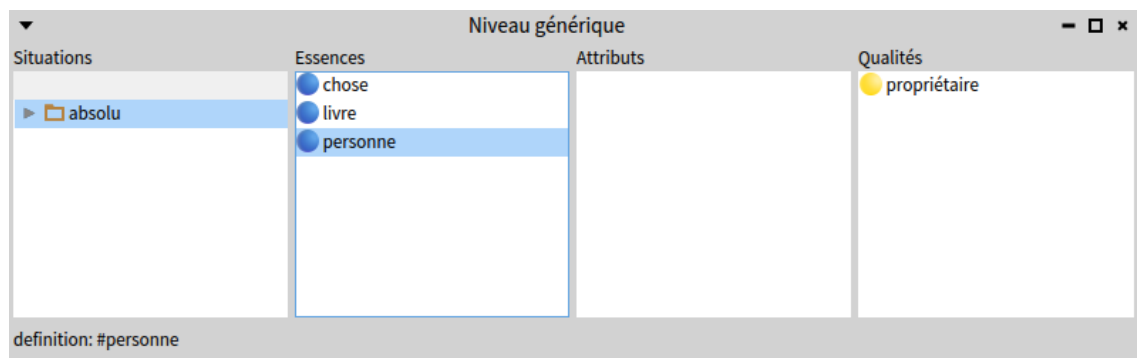
```

iceo definition: #personne.
iceo definition: #livre.
iceo definitionSituation: #possession.
iceo definitionQualite: #propriétaire situation: possession.
iceo definitionQualite: #bien situation: possession.
(possession get: #propriétaire) associationQualite: (possession get:#bien).
personne peutEtre: (possession get: #propriétaire).livre peutEtre: (
    possession get: #bien).
iceo soit: #Paul essence: personne.
iceo soit: 'Pharo by example' essence: livre.
iceo soit: 'possession de Paul' situationGenerique: possession.
(monde get: #Paul) affecteEtat: (possession get: #propriétaire)
    dansSituation: (monde getSituation: 'possession de Paul').
(monde get: 'Pharo by example') affecteEtat: (possession get: #bien)
    dansSituation: (monde get: 'possession de Paul') .
((monde get: #Paul) getEtat: #propriétaire dansSituation: (monde get: '
    possession de Paul')) associationEtat: ((monde get: 'Pharo by example')
    getEtat: #bien dansSituation: (monde get: 'possession de Paul')).

```

Le résultat de cette construction apparait dans les fenêtres suivantes :





On voit que l'attribut 'Pharo by example' de Paul lui est conféré par son état de propriétaire (icône violette).

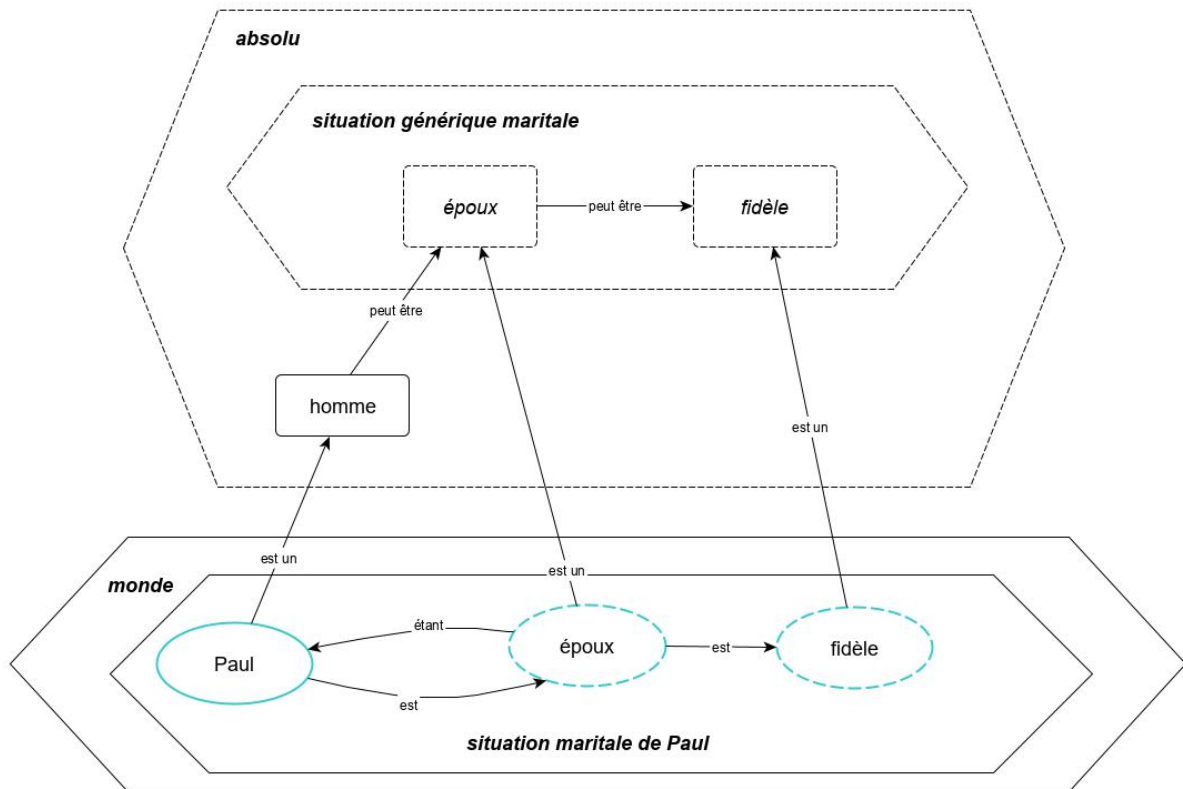
Le fait de dire que le propriétaire d'un livre est une personne (comme nous l'avons fait dans l'exemple 7) est un raccourci de langage.

Bien sûr, un livre ne rentre pas dans la composition d'une personne, et réciproquement !

La même logique s'appliquerait pour dire que Paul a un crayon, une veste, ...

## Exemple 15 : sur la notion d'état dans un état

Considérons le graphe suivant :



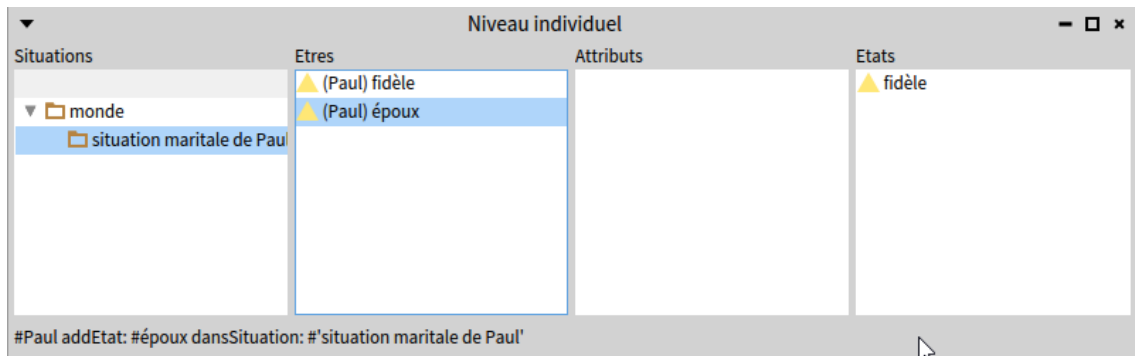
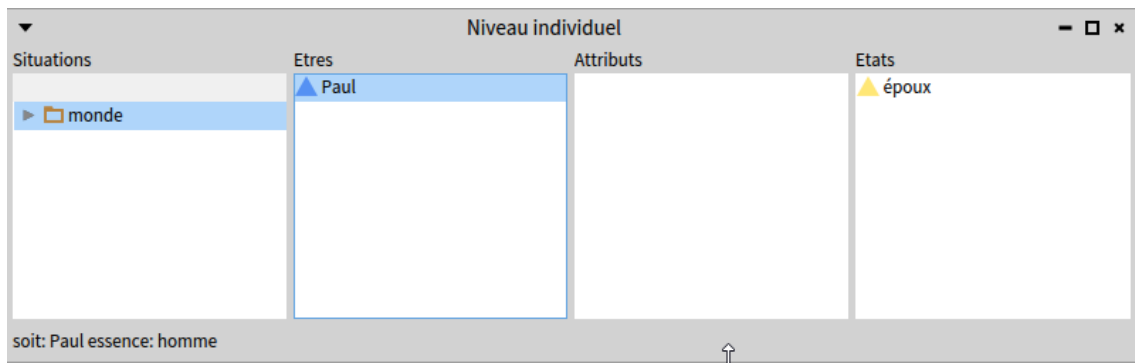
```

iceo definition: #homme.
iceo definitionSituation: 'situation générique maritale'.
iceo definitionQualite: #époux situation:(absolu getSituation: 'situation g
    énérique maritale').
iceo definitionQualite: #fidèle situation: (absolu getSituation:      '
    situation générique maritale').
homme peutEtre: ((absolu getSituation: 'situation générique maritale') get:
    #époux).
iceo soit: 'situation maritale de Paul' situationGenerique: (absolu
    getSituation: 'situation générique maritale').
iceo soit: #Paul essence: homme.
(monde get: #Paul) affecteEtat: ((absolu getSituation: 'situation générique
    maritale') get: #époux) dansSituation: (monde get: 'situation maritale
    de Paul').
((monde get: #Paul) getEtat: #époux dansSituation: (monde get: 'situation
    maritale de Paul')) affecteEtat: ((absolu getSituation: 'situation géné
    rique maritale') get: #fidèle) dansSituation: (monde get: 'situation
    maritale dePaul').
    
```

L'expression "((monde get: #Paul) getEtat: #époux dansSituation: (monde get: 'situation maritale de Paul')) getEtats "

donne : an OrderedCollection(fidèle)

Ceci est visualisé dans les fenêtres suivantes :

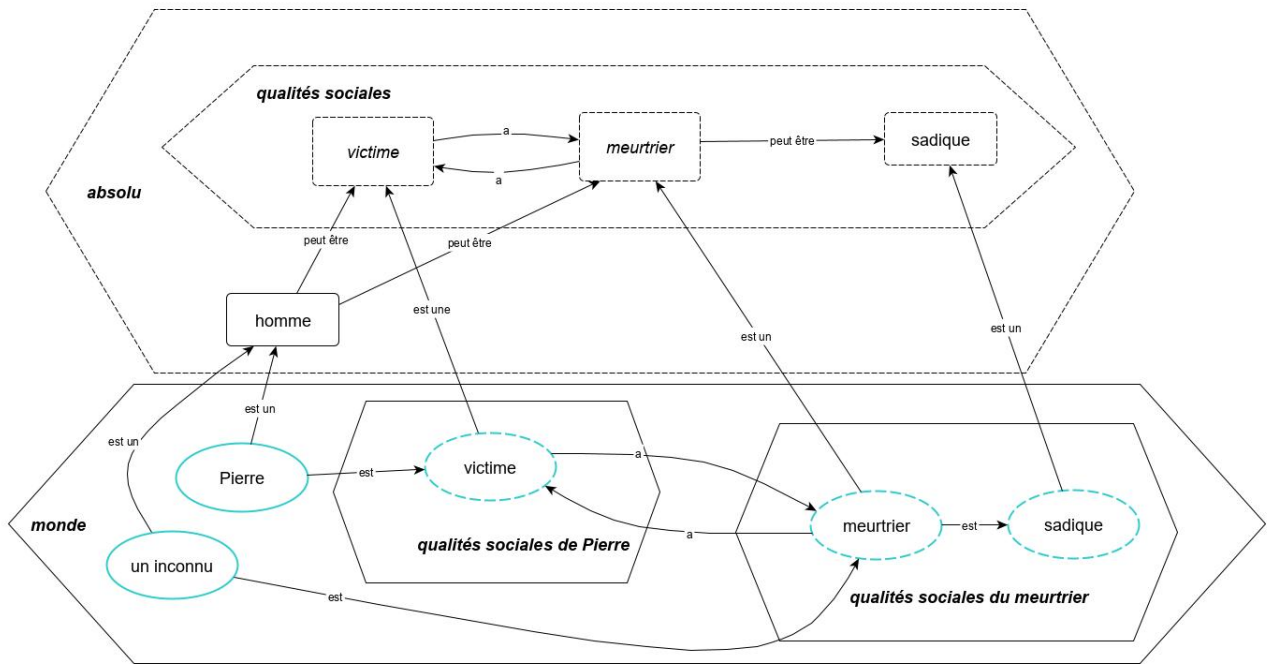


Noter que l'expression " ((monde get: #Paul) getEtat: #fidèle dansSituation: (monde get: 'situation maritale de Paul')) "

ne donnerait rien, car si Paul est fidèle en tant qu'époux, il ne l'est pas forcément dans toutes les situations !

## Exemple 16 : sur la notion d'être inconnu

Considérons le graphe suivant :



Il correspond à la représentation de la phrase : "Le meurtrier de Pierre est sadique".

Cette affirmation n'implique pas la connaissance de l'identité du meurtrier par celui qui la prononce. Nous admettrons que cette phrase a un sens, bien que le meurtrier de Pierre puisse être inconnu; elle peut être formulée en voyant simplement l'état épouvantable de la victime Pierre.

Sa représentation dans ICEO est la suivante :

```

iceo definition: #personne.
iceo definition: #homme genus: personne.
iceo definitionSituation: 'qualités sociales'
iceo definitionQualite: #meurtrier situation: (absolu getSituation: #'qualités sociales').
iceo definitionQualite: #victime situation: (absolu getSituation: #'qualités sociales').
iceo definitionQualite: #sadique situation: (absolu getSituation: #'qualités sociales').
homme peutEtre: ((absolu getSituation: #'qualités sociales') get: #meurtrier).
homme peutEtre: ((absolu getSituation: #'qualités sociales') get: #victime).
((absolu getSituation: #'qualités sociales') get: #victime)
  associationQualite: ((absolu getSituation: #'qualités sociales') get: #meurtrier).
((absolu getSituation: #'qualités sociales') get: #meurtrier) peutEtre: ((absolu getSituation: #'qualités sociales') get: #sadique).
iceo soit: #Pierre essence: homme.
iceo soit: 'qualités sociales de Pierre' situationGenerique: (absolu getSituation: #'qualités sociales').
(monde get: #Pierre) affecteEtat: ((absolu getSituation: #'qualités sociales') get: #victime) dansSituation: (monde get: #'qualités sociales de Pierre').
    
```

```

iceo soit: 'qualités sociales du meurtrier' situationGenerique: (absolu
  getSituation: #'qualités sociales').
" Création d'un état dont l'étant est inconnu "
iceo soitEtat: #meurtrier essence: ((absolu getSituation: #'qualités
  sociales') get: #meurtrier) situationIndividuelle: (monde get: #'
  qualités sociales du meurtrier').
((monde get: #'qualités sociales du meurtrier') get: #meurtrier)
  affecteEtat: ((absolu getSituation: #'qualités sociales') get: #sadique)
  dansSituation: (monde get: #'qualités sociales du meurtrier').
((monde get: #'qualités sociales du meurtrier') get: #meurtrier)
  associationEtat: ((monde get: #'qualités sociales de Pierre') get:
    #victime).

```

```

"((monde get: 'qualités sociales du meurtrier') get: #meurtrier) getEtats"
donne : an OrderedCollection(sadique)

```

```

" (monde get: #Pierre) getEtats " donne : an OrderedCollection(victime)

```

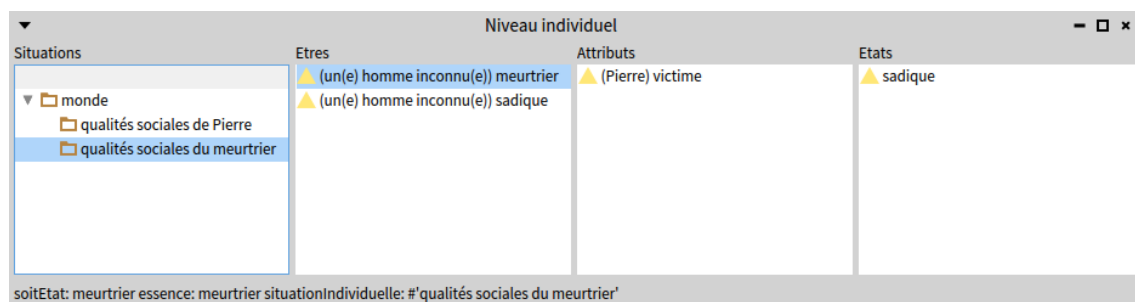
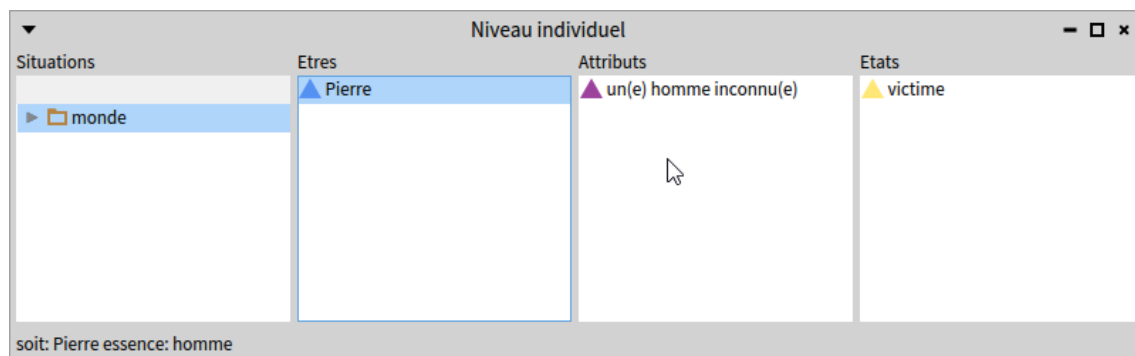
Enfin, l'expression

```

"((monde get: 'qualités sociales du meurtrier') get: #meurtrier) getEtant"
donne : 'un(e) homme inconnu(e)'

```

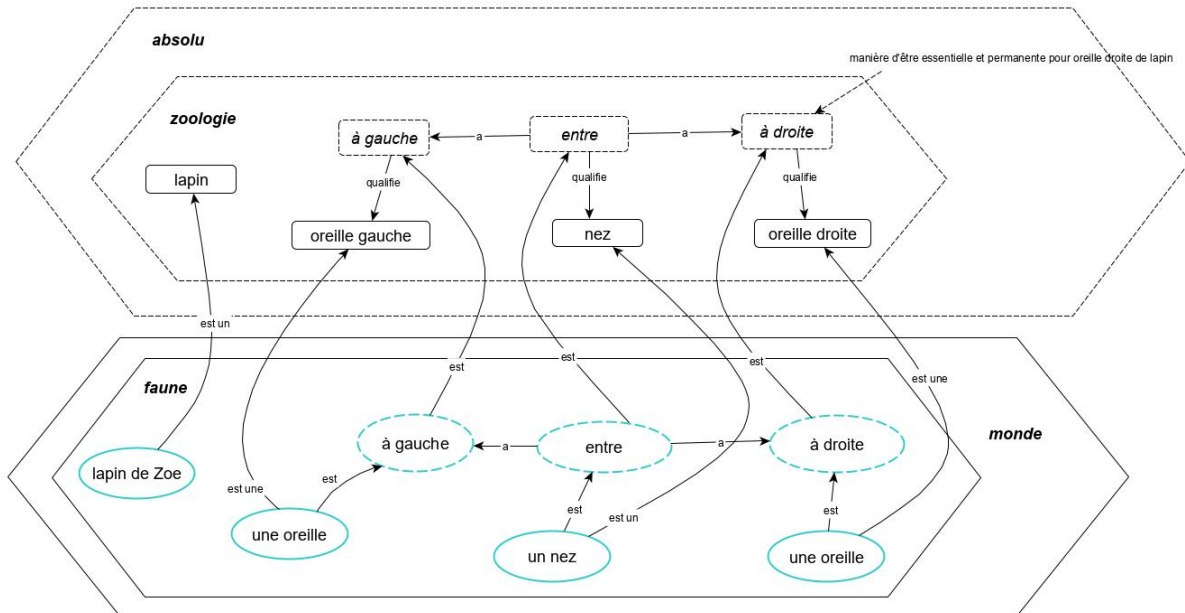
Ceci est visualisé dans les fenêtres suivantes :



Si plusieurs essences avaient la qualité de meurtrier, c'est leur premier genus commun qui aurait été choisi comme essence du meurtrier. Si ce genus était l'essence chose, le meurtrier aurait été simplement identifié comme étant "quelque chose".

## Exemple 17 : sur la notion de contrainte structurelle interne

Le graphe suivant exprime que le nez d'un lapin doit se situer entre ses deux oreilles :



(Pour ne pas complexifier l'aspect visuel de ce diagramme, le fait que le nez et les oreilles sont des attributs du lapin n'a pas été représenté).

Ce graphe exprime que les manières d'être "à gauche", "entre" et "à droite" sont essentielles et permanentes pour les essences oreille gauche, nez et oreille droite de lapin.

La représentation dans ICEO est la suivante :

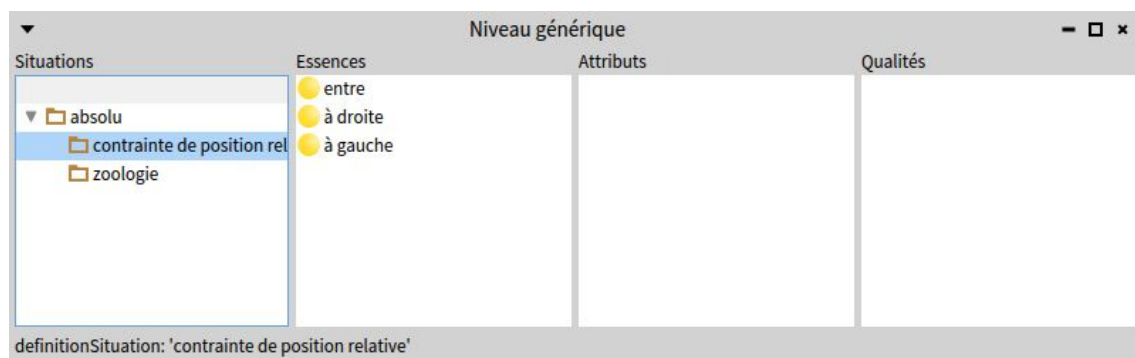
```
iceo definitionSituation: #zoologie.
iceo definition: #lapin situation: zoologie.
iceo definitionAttribut: #nez de: (zoologie get: #lapin) cardinalite: 1.
iceo definitionAttribut: 'oreille gauche' de: (zoologie get: #lapin)
  cardinalite: 1.
iceo definitionAttribut: 'oreille droite' de: (zoologie get: #lapin)
  cardinalite: 1.
iceo definitionQualiteEssentielle: 'entre' genus: ((absolu getSituation:
  'contrainte de position relative') get: 'entre') pour: ((zoologie get
  : #lapin) getEssenceAttribut: #nez) effectivite: #permanente.
iceo definitionQualiteEssentielle: 'à gauche' genus: ((absolu getSituation:
  'contrainte de position relative') get: 'à gauche') pour: ((zoologie
  get: #lapin) getEssenceAttribut: 'oreille gauche') effectivite: #
  permanente.
iceo definitionQualiteEssentielle: 'à droite' genus: ((absolu getSituation:
  'contrainte de position relative') get: 'à droite') pour: ((zoologie
  get: #lapin) getEssenceAttribut: 'oreille droite') effectivite: #
  permanente.
(((zoologie get: #lapin) getEssenceAttribut: #nez) getQualite: 'entre')
  associationQualite: (((zoologie get: #lapin) getEssenceAttribut: '
  oreille gauche') getQualite: 'à gauche').
(((zoologie get: #lapin) getEssenceAttribut: #nez) getQualite: 'entre')
  associationQualite: (((zoologie get: #lapin) getEssenceAttribut: 'oreille
```

```

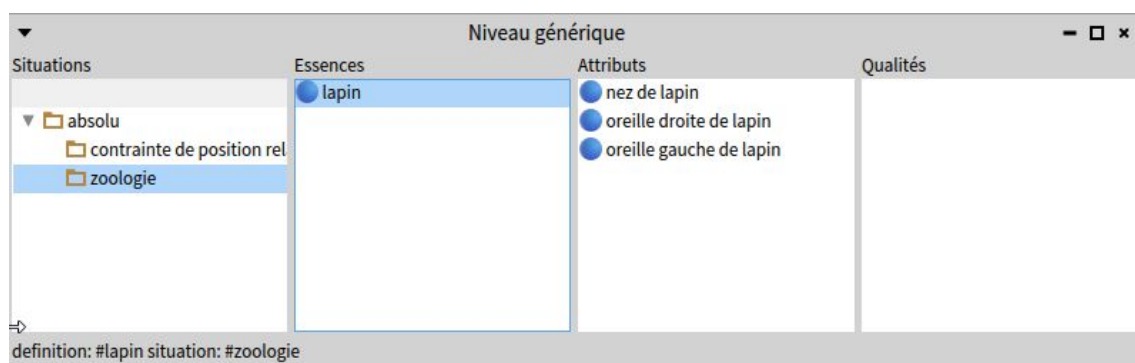
    droite ') getQualite: 'à droite ').
iceo soit: #faune situationGenerique: zoologie
iceo soit: 'Lapin de Zoé' essence: (zoologie get: #lapin)
    situationIndividuelle: (monde get: #faune)
(((monde get: 'Lapin de Zoé') getEtreAttribut: #nez) getEtat: #entre)
    associationEtat: (((monde get: 'Lapin de Zoé') getEtreAttribut: 'oreille
    gauche') getEtat: 'à gauche')
(((monde get: 'Lapin de Zoé') getEtreAttribut: #nez) getEtat: #entre)
    associationEtat: (((monde get: 'Lapin de Zoé') getEtreAttribut: 'oreille
    droite') getEtat: 'à droite ')

```

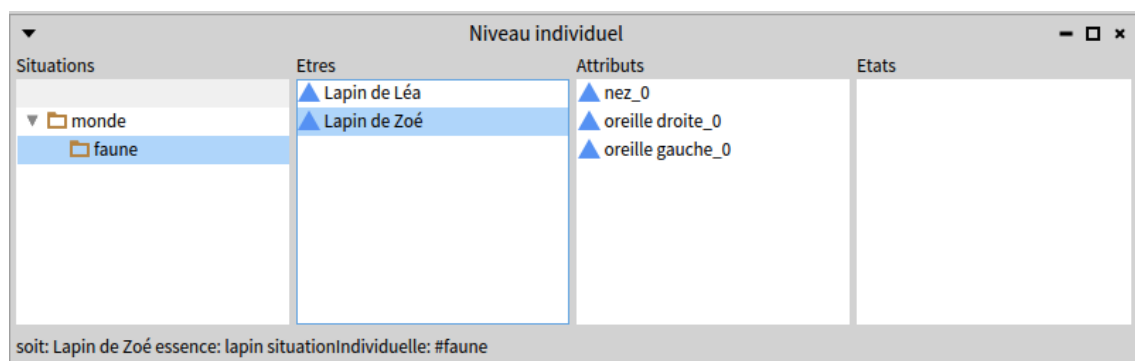
Un SgBrowser visualise les manières d'être définies dans la situation générique "contrainte de position relative" :



L'essence lapin est définie dans la situation zoologie :



Voici une visualisation du lapin de Zoé :

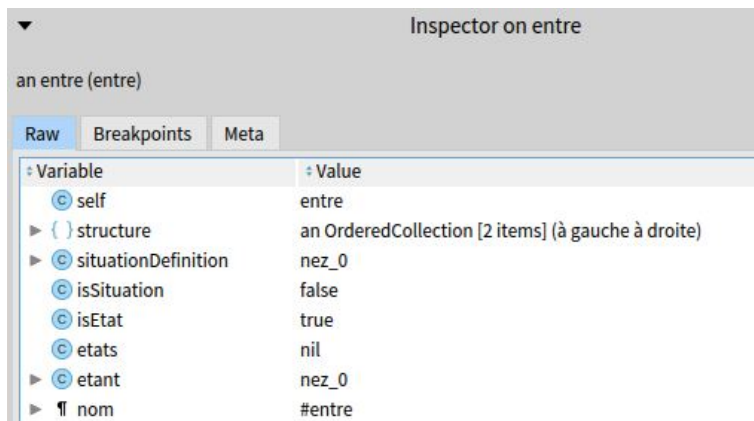


Le nez du lapin de Zoé a été automatiquement instancié, comme ses oreilles, car leur cardinalité au niveau de l'essence lapin est de 1.

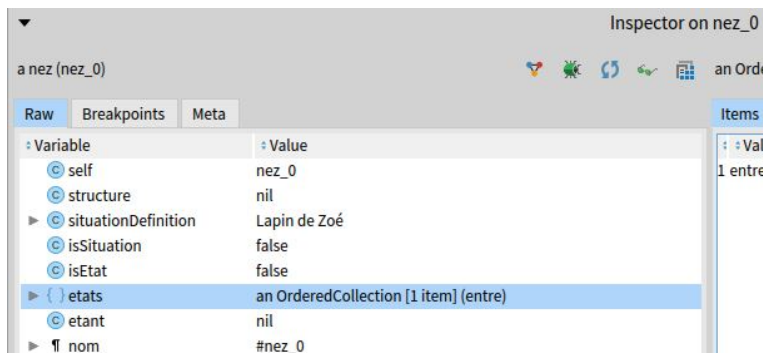
L'inspection de

```
"((monde get: #faune) get: 'Lapin de Zoé') getEtreAttribut: #nez) getEtats at: 1 "
```

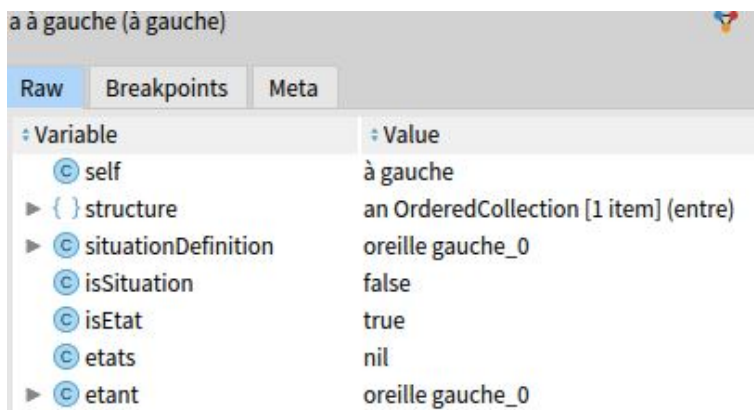
donne :



Faisons l'inspection du nez :



L'inspection de l'état "à gauche" donne :



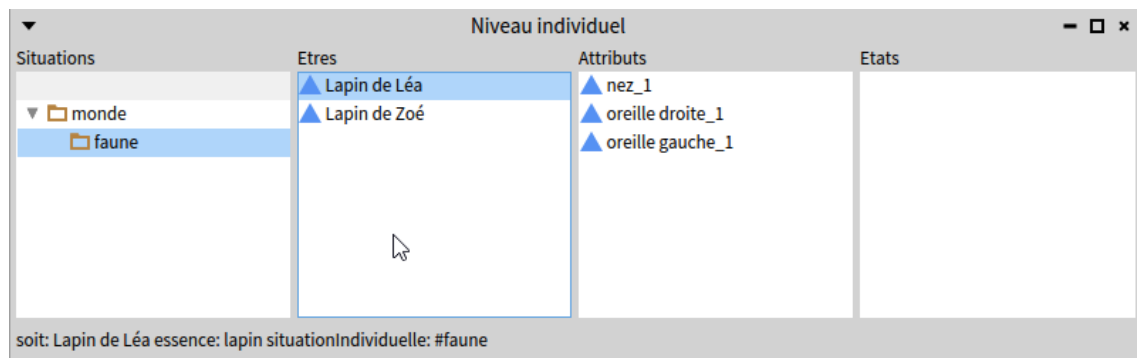
On constate qu'il s'agit bien de l'état de l'oreille gauche du lapin de Zoé.

Créons maintenant un lapin nommé Léa :

```
iceo soit: 'Lapin de Léa' essence: (zoologie get: #lapin)
  situationIndividuelle: (monde get: #faune)
  ((monde get: 'Lapin de Léa') getEtreAttribut: #nez) getEtat: #entre)
  associationEtat: (((monde get: 'Lapin de Léa') getEtreAttribut: 'oreille
    gauche') getEtat: #'à gauche')
  ((monde get: 'Lapin de Léa') getEtreAttribut: #nez) getEtat: #entre)
```



```
associationEtat: (((monde get: 'Lapin de Léa') getEtreAttribut: 'oreille
droite ') getEtat: #'à droite ')
```



On constate que le nez et les oreilles des deux lapins sont différents, ce qui est logique.

Mais qu'en est-il de leurs états ? Faisons un test d'identité en évaluant l'expression

```
" (((monde get: 'Lapin de Zoé') getEtreAttribut: #'oreille gauche') getEtats
at: 1) == ((monde get: 'Lapin de Léa') getEtreAttribut: #'oreille gauche')
getEtats at: 1) "
```

Le résultat est : false

Ce résultat est tout-à-fait dans la logique d'ICEO.

En effet, il est important de distinguer l'état "à gauche" de l'oreille gauche du lapin de Léa de l'état "à gauche" de l'oreille gauche du lapin de Zoé.

Sachant que, comme nous l'avons vu, un état peut lui même avoir des états, il serait possible par exemple que l'un soit "un peu plus à gauche" que l'autre.

## Exemple 18 : sur la notion de famille

Cet exemple repart de l'exemple 13 sur la subsumption des manières d'être pour représenter le famille Gonthier où les parents Charles et Hermeline possèdent deux filles Capucine et Juliette et deux fils Martin et Jean :

```
iceo definition: #personne.
iceo definition: #femme genus: personne.
iceo definition: #homme genus: personne.
iceo definitionSituation: 'famille '.
iceo definitionQualite: #parent situation: (absolu getSituation: #famille).
iceo definitionQualite: #enfant situation: (absolu getSituation: #famille).
iceo definitionQualite: #soeur situation: (absolu getSituation: #famille).
iceo definitionQualite: #frère situation: (absolu getSituation: #famille).
iceo definitionQualite: #fils situation: (absolu getSituation: #famille)
    genus: ((absolu getSituation: #famille) get: #enfant).
iceo definitionQualite: #père situation: (absolu getSituation: #famille)
    genus: ((absolu getSituation: #famille) get: #parent).
iceo definitionQualite: #fille situation: (absolu getSituation: #famille)
    genus: ((absolu getSituation: #famille) get: #enfant).
iceo definitionQualite: #mère situation: (absolu getSituation: #famille)
    genus: ((absolu getSituation: #famille) get: #parent).
((absolu getSituation: #famille) get: #parent) associationQualite: ((absolu
    getSituation: #famille) get: #enfant).
((absolu getSituation: #famille) get: #père) associationQualite: ((absolu
    getSituation: #famille) get: #fils).
((absolu getSituation: #famille) get: #père) associationQualite: ((absolu
    getSituation: #famille) get: #fille).
((absolu getSituation: #famille) get: #mère) associationQualite: ((absolu
    getSituation: #famille) get: #fils).
((absolu getSituation: #famille) get: #mère) associationQualite: ((absolu
    getSituation: #famille) get: #fille).
((absolu getSituation: #famille) get: #frère) associationQualite: ((absolu
    getSituation: #famille) get: #soeur).
((absolu getSituation: #famille) get: #frère) associationQualite: ((absolu
    getSituation: #famille) get: #frère).
((absolu getSituation: #famille) get: #soeur) associationQualite: ((absolu
    getSituation: #famille) get: #soeur).
homme peutEtre: ((absolu getSituation: #famille) get: #père).
homme peutEtre: ((absolu getSituation: #famille) get: #fils).
femme peutEtre: ((absolu getSituation: #famille) get: #mère).
femme peutEtre: ((absolu getSituation: #famille) get: #fille).
((absolu getSituation: #famille) get: #fille) peutEtre: ((absolu
    getSituation: #famille) get: #soeur).
((absolu getSituation: #famille) get: #fils) peutEtre: ((absolu
    getSituation: #famille) get: #frère).
```

Au niveau générique, rien n'a changé par rapport à l'exemple 13.

Au niveau individuel commençons par créer les deux époux Charles et Hermeline :

```
iceo soit: #Gonthier situationGenerique: famille.
iceo soit: #Hermeline essence: femme.
iceo soit: #Charles essence: homme.
(monde get: #Hermeline) affecteEtat: (famille get: #épouse) dansSituation:
    (monde getSituation: #Gonthier).
(monde get: #Charles) affecteEtat: (famille get: #époux) dansSituation: (
    monde getSituation: #Gonthier).
```

```
((monde get: #Charles) getEtat: #époux) associationEtat: ((monde get: #
Hermeline) getEtat: #épouse dansSituation: (monde getSituation: #
Gonthier)).
```

ICEO étant implanté en Smalltalk, il est possible de mixer du code ICEO avec du code Smalltalk.

Nous en avons profité pour écrire une méthode de l'essence femme qui correspond à la mise au monde d'un enfant :

```
femme compile: ' metAuMonde: prénom famille: uneFamille essence: uneEssence
| époux |
iceo soit: prénom essence: uneEssence.
(self getEtat: #épouse) notNil ifTrue: [époux := ((self getEtresAttributsQuiSont: (famille get: #époux))) at: 1].
(self getEtat: #mère) ifNil: [self affecteEtat: (famille get: #mère) dansSituation: uneFamille].
(époux getEtat: #père) ifNil: [époux affecteEtat: (famille get: #père) dansSituation: uneFamille].
uneEssence getNom == #homme
ifTrue: [ (monde get: prénom) affecteEtat: (famille get: #fils) dansSituation: uneFamille.
(self getEtat: #mère) associationEtat: ((monde get: prénom) getEtat: #fils dansSituation: uneFamille).
(époux getEtat: #père) associationEtat: ((monde get: prénom) getEtat: #fils dansSituation: uneFamille).

"Si la femme a déjà des enfants, le dernier né est leur frère"
"I"étant de ce frère est l"état fils du nouveau né"
"I"état frère de l"état fils du nouveau né sera ici le même pour tous les autres enfants"
((self getEtresAttributsQuiSont: (famille get: #enfant)) size > 0) ifTrue: [
((monde get: prénom) getEtat: #frère) ifNil: [ ((monde get: prénom) getEtat: #fils) affecteEtat: (famille get: #frère) dansSituation: uneFamille]].
((self getEtresAttributsQuiSont: (famille get: #enfant)) copyWithout: (monde get: prénom)) do: [:each |
(each getEtat: #fille) ifNotNil: [
((each getEtat: #fille) getEtat: #soeur) ifNil: [(each getEtat: #fille) affecteEtat: (famille get: #soeur) dansSituation: uneFamille].
((each getEtat: #fille) getEtat: #soeur) associationEtat: (((monde get: prénom) getEtat: #fils) getEtat: #frère)].
(each getEtat: #fils) ifNotNil: [
((each getEtat: #fils) getEtat: #frère) ifNil: [(each getEtat: #fils) affecteEtat: (famille get: #frère) dansSituation: uneFamille].
((each getEtat: #fils) getEtat: #frère) associationEtat: (((monde get: prénom) getEtat: #fils) getEtat: #frère)]]]
ifFalse: [ (monde get: prénom) affecteEtat: (famille get: #fille) dansSituation: uneFamille.
(self getEtat: #mère) associationEtat: ((monde get: prénom) getEtat: #fille dansSituation: uneFamille).
(époux getEtat: #père) associationEtat: ((monde get: prénom) getEtat: #fille dansSituation: uneFamille).

"Si la femme a déjà des enfants, le dernier né est leur soeur"
"I"étant de cette soeur est l"état fille du nouveau né"
"I"état soeur de l"état fille du nouveau né sera ici le même pour tous les autres enfants"
((self getEtresAttributsQuiSont: (famille get: #enfant)) size > 0) ifTrue: [
((monde get: prénom) getEtat: #soeur) ifNil: [ ((monde get: prénom) getEtat: #fille) affecteEtat: (famille get: #soeur) dansSituation: uneFamille]].
((self getEtresAttributsQuiSont: (famille get: #enfant)) copyWithout: (monde get: prénom)) do: [:each |
(each getEtat: #fille) ifNotNil: [
((each getEtat: #fille) getEtat: #soeur) ifNil: [(each getEtat: #fille) affecteEtat: (famille get: #soeur) dansSituation: uneFamille].
((each getEtat: #fille) getEtat: #soeur) associationEtat: (((monde get: prénom) getEtat: #fille) getEtat: #soeur)].
(each getEtat: #fils) ifNotNil: [
((each getEtat: #fils) getEtat: #frère) ifNil: [(each getEtat: #fils) affecteEtat: (famille get: #frère) dansSituation: uneFamille].
((each getEtat: #fils) getEtat: #frère) associationEtat: (((monde get: prénom) getEtat: #fille) getEtat: #soeur)]]] '.
```

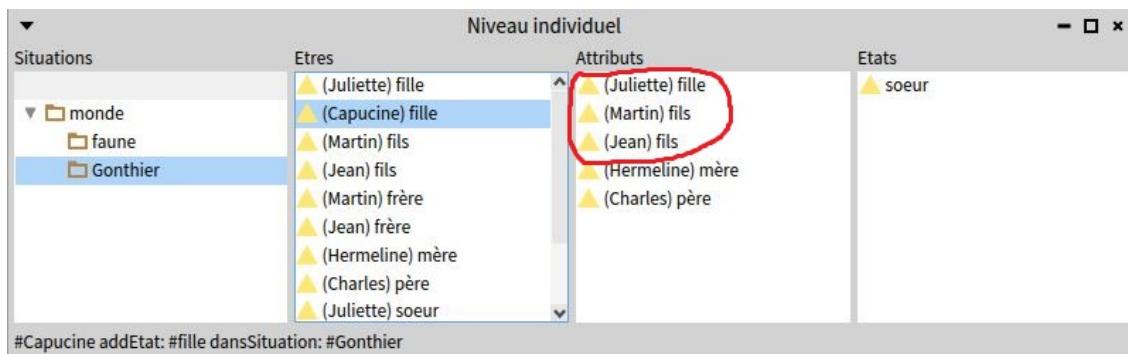
Hermeline met au monde ses quatre enfants :

```
(monde get: #Hermeline) metAuMonde: #Capucine famille: (monde get: #
Gonthier) essence: femme.
(monde get: #Hermeline) metAuMonde: #Jean famille: (monde get: #Gonthier)
essence: homme.
(monde get: #Hermeline) metAuMonde: #Martin famille: (monde get: #Gonthier)
essence: homme.
(monde get: #Hermeline) metAuMonde: #Juliette famille: (monde get: #
Gonthier) essence: femme.
```

Au niveau individuel, Capucine est une fille :



En tant que fille, elle est soeur de Juliette, Jean et Martin :



Nous en avons surchargé la méthode "printOn: aStream" de la classe Object de Smalltalk au niveau de l'essence chose, pour afficher le nom des membres d'une famille avec leur nom de famille :

```
chose compile: ' printOn: aStream
self isEtat
  ifTrue: [self getEtats
    detect: [:e | e getSituationDefinition class getNom == #famille]
    ifFound: [:x | aStream nextPutAll: self getEtat getNom, " ", x getSituationDefinition getNom]
    ifNone: [aStream nextPutAll: self getNom]]
  ifFalse: [self getEtats
    detect: [:e | e getSituationDefinition class getNom == #famille]
    ifFound: [:x | aStream nextPutAll: self getNom, " ", x getSituationDefinition getNom]
    ifNone: [aStream nextPutAll: self getNom]]'
```

Ainsi, voici les réponses données par diverses expressions :

```
((monde get: #Charles) getEtresAttributsQuiSont: ((absolu getSituation:
#famille) get: #enfant))
```

```
donne : an OrderedCollection(Capucine Gonthier Martin Gonthier Jean Gonthier Juliette
Gonthier)
```

```
((monde get: #Charles) getEtresAttributsQuiSont: ((absolu getSituation:
#famille) get: #fille))
```

```
donne : an OrderedCollection(Capucine Gonthier Juliette Gonthier)
```

```
((monde get: #Capucine) getEtresAttributsQuiSont: ((absolu getSituation:
```

```
#famille) get: #frère)) donne : an OrderedCollection(Martin Gonthier Jean Gonthier)
```

```
((monde get: #Martin) getEtresAttributsQuiSont: ((absolu getSituation: #famille) get: #soeur))
```

```
donne : an OrderedCollection(Capucine Gonthier Juliette Gonthier)
```

```
((monde get: #Martin) getEtresAttributsQuiSont: ((absolu getSituation: #famille) get: #frère))
```

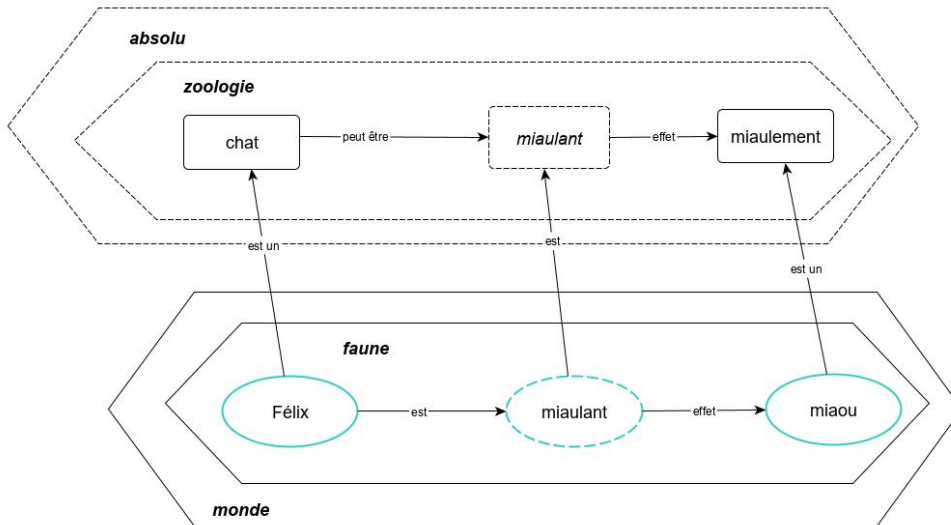
```
donne : an OrderedCollection(Jean Gonthier)
```

Noter que si dans cet exemple Capucine possède, en tant que soeur, une soeur et deux frères, il ne serait pas possible de préciser par exemple qu'elle est une gentille soeur pour Juliette et une soeur autoritaire par rapport à Martin.

Pour cela, il faudrait procéder comme nous l'avons fait dans l'exemple 14\_4 pour différencier les états d'époux de Paul par rapport à ses deux épouses Juliette et Yvette.

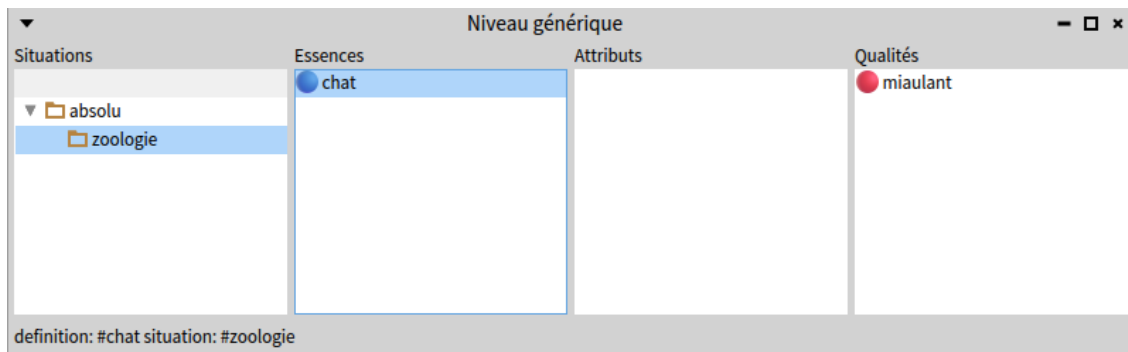
## Exemple 19 : sur la notion d'action

Considérons le graphe suivant :



```
iceo definitionSituation: #zoologie.
iceo definition: #chat situation: zoologie.
iceo definitionQualiteEssentielle: #miaulant pour: (zoologie
get: #chat) effectivite: #intermittente.
iceo definitionAttribut:#miaulement de: ((zoologie get: #chat) getQualite:
#miaulant).
```

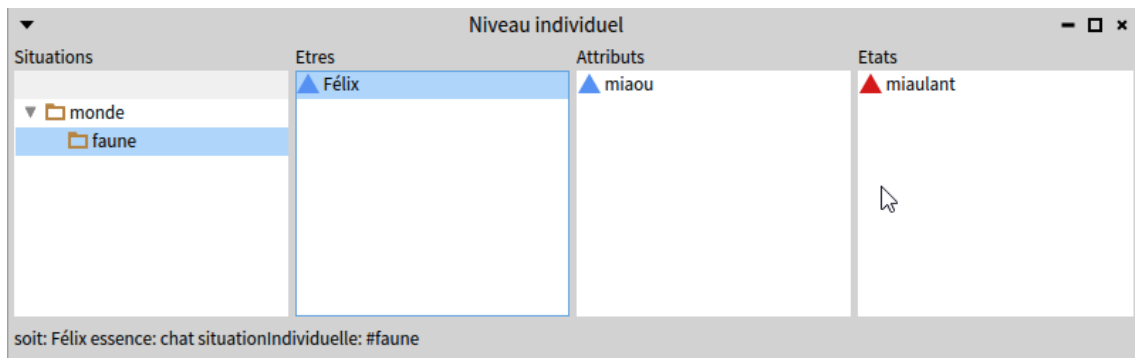
Dans cet exemple, la qualité "miaulant" de chat est essentielle (le chat peut miauler par essence)



mais son effectivité est intermittente (un chat ne miaule pas sans arrêt).

Il est donc nécessaire de préciser que Félix est ici en train de miauler.

```
iceo soit: #faune situationGenerique: zoologie
iceo soit: #Félix essence: (zoologie get: #chat)
(((monde get: #faune) get: #Félix) affecteEtatEssentiel: ((zoologie get: #
chat) getQualite: #miaulant)
(((monde get: #faune) get: #Félix) getEtat: #miaulant) attributionEtre: #
miaou essence: (((((zoologie get: #chat) getQualite: #miaulant))
getEssenceAttribut: #miaulement)
```



Supposons que la méthode Smalltalk suivante soit associée à la manière d'être miaulant :

```
((zoologie get: #chat) getQualite: #miaulant) compile: 'miaule .  
  ^ (self getEtresAttributs asOrderedCollection at: 1) getNom'.
```

L'envoi du message "miaule" à l'état miaulant de Félix par :

```
((monde get: #faune) get: #Félix) getEtat: #miaulant) miaule  
donne : Miaou
```

Supposons que Maya soit une chatte dont le miaulement produit "meow" :

```
iceo soit: #Maya essence: (zoologie get: #chat)  
((monde get: #faune) get: #Maya) affecteEtatEssentiel: ((zoologie get: #  
  chat) getQualite: #miaulant)  
(((monde get: #faune) get: #Maya) getEtat: #miaulant) attributionEtre: #  
  meow essence: (((zoologie get: #chat) getQualite: #miaulant))  
  getEssenceAttribut: #miaulement)
```

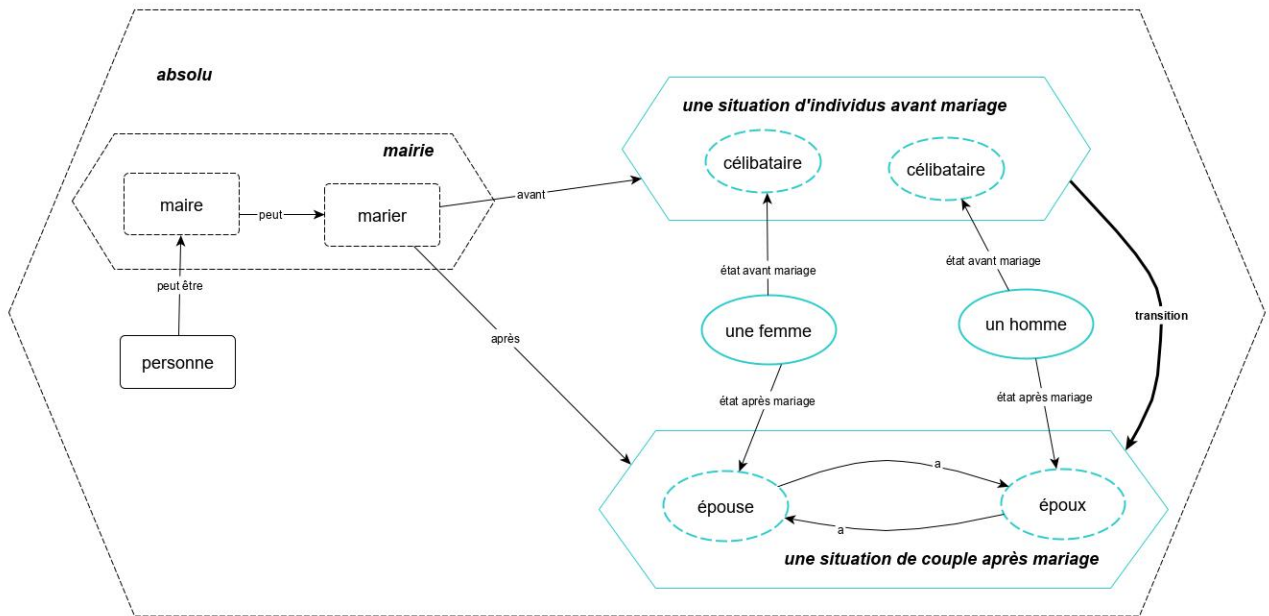
L'envoi du message miaule à l'état miaulant de Maya par :

```
((monde get: #faune) get: #Maya) getEtat: #miaulant) miaule  
donne : Meow
```

## Exemple 20 : sur les changements de situation

Pour cet exemple, nous allons repartir de l'exemple sur les relations entre manières d'être.

Le graphe suivant décrit le changement de situation d'une femme et d'un homme consécutif à leur mariage par une personne ayant la qualité de maire :



La représentation dans ICEO est la suivante :

```
iceo definition: #personne.
iceo definition: #femme genus: personne.
iceo definition: #homme genus: personne.
iceo definitionSituation: 'situation de couple '.
iceo definitionSituation: 'situation d'individu '.
iceo definitionSituation: #mairie.
iceo definitionQualite: #épouse situation: (absolu getSituation: 'situation
de couple ').
iceo definitionQualite: #époux situation: (absolu getSituation: 'situation
de couple ').
iceo definitionQualite: #célibataire situation: (absolu getSituation: '
situation d'individu ').
iceo definitionQualite: #maire situation: (absolu getSituation: #mairie).
iceo definitionQualiteEssentielle: #mariant pour: ((absolu getSituation:
#mairie) get: #maire) effectivite: #intermittente.
((absolu getSituation: 'situation de couple ') get: #épouse)
associationQualite: ((absolu getSituation: 'situation de couple ')
get: #époux).
femme peutEtre: ((absolu getSituation: 'situation de couple ') get: #épouse).
femme peutEtre: ((absolu getSituation: 'situation d'individu ')
get: #célibataire).
homme peutEtre: ((absolu getSituation: 'situation de couple ') get: #époux).
homme peutEtre: ((absolu getSituation: 'situation d'individu ')
get: #célibataire).
personne peutEtre: ((absolu getSituation: #mairie) get: #maire).
```

Supposons que la méthode suivante soit associée à la manière d'être mariant de la manière d'être maire :



```

(((absolu getSituation: #mairie) get: #maire) getQualite: #mariant) compile: 'marie: s1 to: s2
| unHomme uneFemme |
s1 getElements
do: [:etat |
    etat getEssence getNom == #célibataire
    ifTrue: [etat getEtat removeEtat: etat].
    etat getEtat getEssence getNom == #homme
    ifTrue: [unHomme := etat getEtat.
        unHomme
        affecteEtat: (s2 class get: #époux)
        dansSituation: s2]
    ifFalse: [uneFemme := etat getEtat.
        uneFemme
        affecteEtat: (s2 class get: #épouse)
        dansSituation: s2]].
(unHomme getEtat: #époux dansSituation: s2)
associationEtat: (uneFemme getEtat: #épouse dansSituation: s2)'.

```

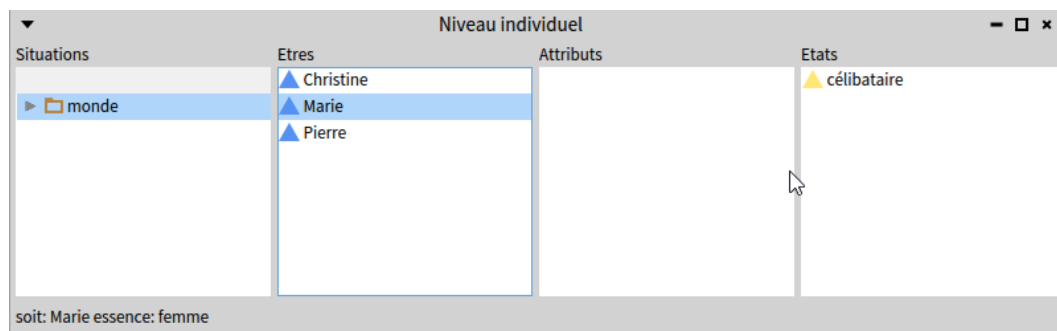
Christine, Marie et Pierre sont créés de la manière suivante :

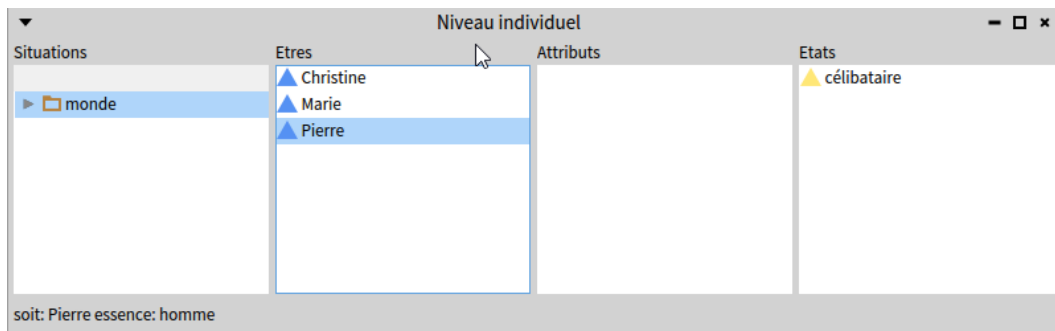
```

iceo soit: #Marie essence: femme.
iceo soit: #Pierre essence: homme.
iceo soit: #Christine essence: personne.
iceo soit: 'une mairie' situationGenerique: (absolu getSituation: #mairie).
(monde get: #Christine) affecteEtat: ((absolu getSituation: #mairie) get: #
    maire) dansSituation: (monde getSituation: 'une mairie').
(((monde get: #Christine) getEtat: #maire) affecteEtatEssentiel: (((absolu
    getSituation: #mairie) get: #maire) getQualite: #mariant)).
iceo soit: 'situation de Pierre et Marie avant mariage' situationGenerique:
    (absolu getSituation: 'situation d''individu').
iceo soit: 'situation de Pierre et Marie après mariage' situationGenerique:
    (absolu getSituation: 'situation de couple').
(monde get: #Marie) affecteEtat: ((absolu getSituation: 'situation d'
    individu') get: #célibataire) dansSituation: (monde getSituation: '
    situation de Pierre et Marie avant mariage').
(monde get: #Pierre) affecteEtat: ((absolu getSituation: 'situation d'
    individu') get: #célibataire) dansSituation: (monde getSituation: '
    situation de Pierre et Marie avant mariage').
personne peutEtre: ((absolu getSituation: #mairie) get: #maire).

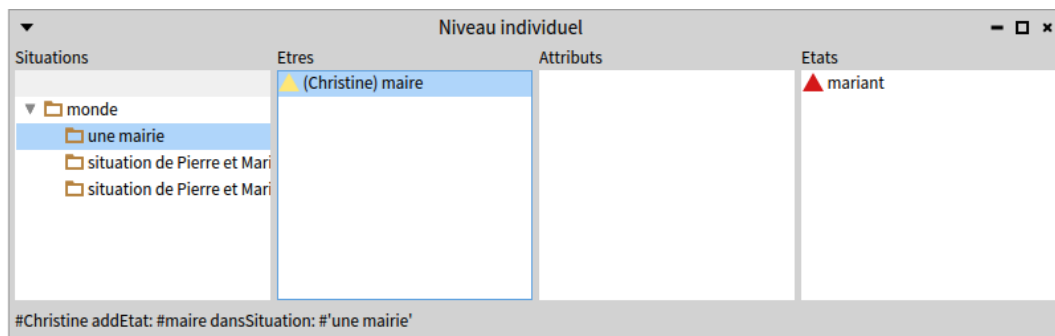
```

Avant leur mariage, Pierre et Marie sont célibataires :





et Christine en tant que maire est dans l'état essentiel et intermittent mariant :

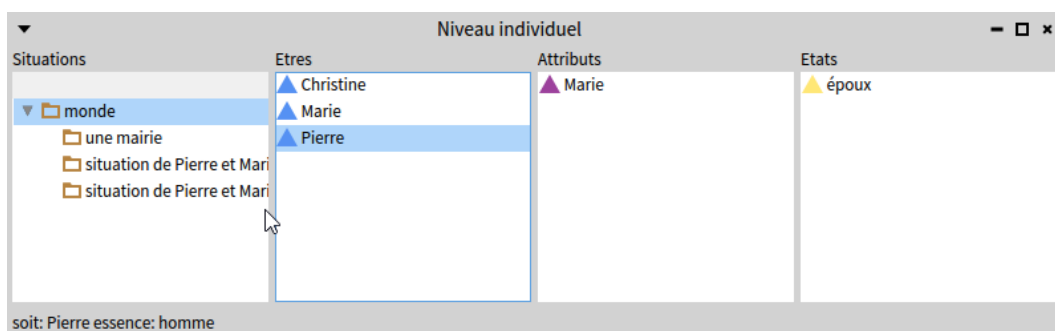
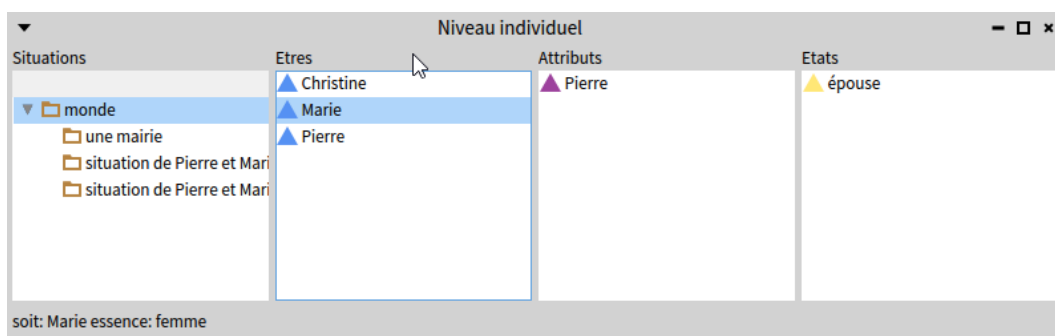


L'action de marier de Christine en tant que maire dans l'état mariant va les faire passer dans une situation de couple, avec les états d'épouse et d'époux :

```
((monde get: #Christine) getEtat: #maire) getEtat: #mariant) marie: (
monde getSituation: 'situation de Pierre et Marie avant mariage') to:
(monde getSituation: 'situation de Pierre et Marie après mariage').
```

Sélectionner cette instruction et faire un "Do it".

Après mariage on constate que Marie et Pierre ont bien été mariés par Christine en tant que maire.



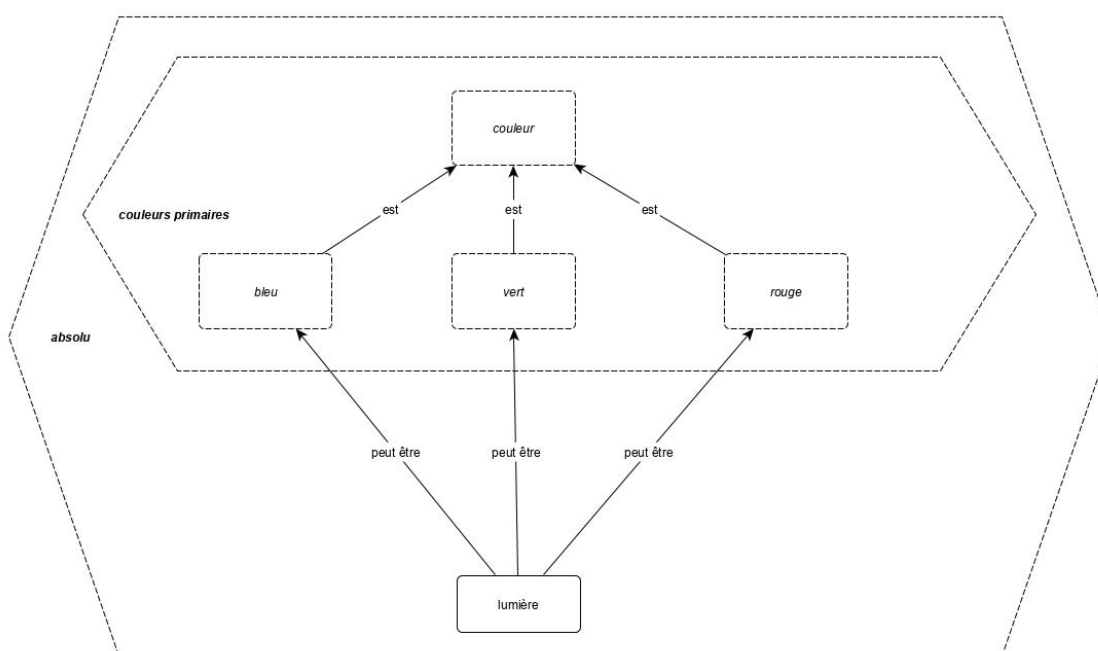
## Exemple 21 : sur la notion de couleur

En soi, la lumière se présente comme de l'eau : elle n'a pas de forme, sa substance est massière.

Une lumière peut avoir différentes qualités. Ainsi une lumière peut être chaude (quand sa couleur tire vers l'orange) ou froide (lorsque sa couleur tire vers le bleu), ..., de la même manière que l'eau peut nous paraître chaude, tiède, froide, ...

Une couleur est une qualité de la lumière perçue par un sujet.

Voici la représentation des trois couleurs dites "primaires" :



Une lumière perçue peut être bleue, rouge ou verte.

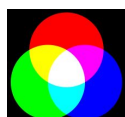
Les couleurs primaires bleu, vert ou rouge sont celles des trois (spectres de) lumières perçues et distinguées par l'œil humain que nous qualifierons de "monochromatiques"

Une lumière peut aussi être composée de lumières.

Les différentes compositions de lumières font apparaître d'autres couleurs possibles, reconnaissables par celui qui a appris à reconnaître ces compositions. Ainsi "blanc" est la couleur d'une lumière composée à parts égales de lumière rouge, verte et bleue.

Les couleurs des lumières composées à parts égales de deux lumières ayant une couleur primaire sont appelées secondaires. Ce sont les couleurs cyan, magenta et jaune.

Nous qualifierons ces lumières de "polychromatiques"



Les couleurs tertiaires viennent de la composition à parts égales d'une lumière ayant une couleur secondaire et de l'une des deux lumières de couleur primaire de sa couleur secondaire. Ainsi la couleur vermillon est composée à partir du jaune et du rouge.

noir est la couleur d'une lumière dont la composition est vide.

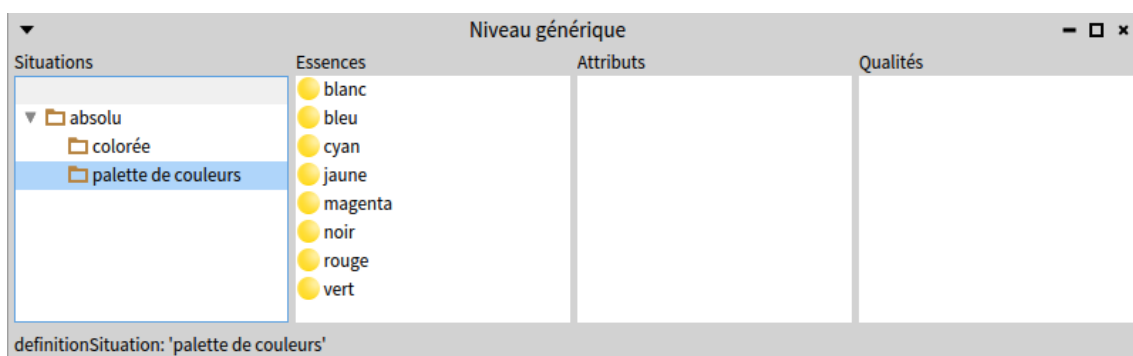
La palette des couleurs primaires et secondaires peut être définie de la manière suivante:

```

iceo definitionQualite: #couleur situation: (absolu getSituation: #
    colorée)
iceo definitionSituation: 'palette de couleurs '
iceo definitionQualite: #bleu situation: (absolu getSituation: '
    palette de couleurs ') genus: ((absolu getSituation: #colorée)
    get: #couleur)
iceo definitionQualite: #vert situation: (absolu getSituation: '
    palette de couleurs ') genus: ((absolu getSituation: #colorée)
    get: #couleur)
iceo definitionQualite: #rouge situation: (absolu getSituation: '
    palette de couleurs ') genus: ((absolu getSituation: #colorée)
    get: #couleur)
iceo definitionQualite: #cyan situation: (absolu getSituation: '
    palette de couleurs ') genus: ((absolu getSituation: #colorée)
    get: #couleur)
iceo definitionQualite: #magenta situation: (absolu getSituation: '
    palette de couleurs ') genus: ((absolu getSituation: #colorée)
    get: #couleur)
iceo definitionQualite: #jaune situation: (absolu getSituation: '
    palette de couleurs ') genus: ((absolu getSituation: #colorée)
    get: #couleur)
iceo definitionQualite: #blanc situation: (absolu getSituation: '
    palette de couleurs ') genus: ((absolu getSituation: #colorée)
    get: #couleur)
iceo definitionQualite: #noir situation: (absolu getSituation: '
    palette de couleurs ') genus: ((absolu getSituation: #colorée)
    get: #couleur)

```

La palette de couleurs ainsi définie est :



Voici comment peuvent se définir les différentes lumières :

```

iceo definition: #lumière.
iceo definition: 'lumière monochromatique' genus: lumière.
iceo definition: 'lumière polychromatique' genus: lumière.
iceo definition: 'lumière bleu' genus: (absolu get: 'lumière
    remonochromatique').
iceo definitionQualiteEssentielle: 'bleu' pour: (absolu get: 'lumière
    re bleu') effectivite: #permanente.

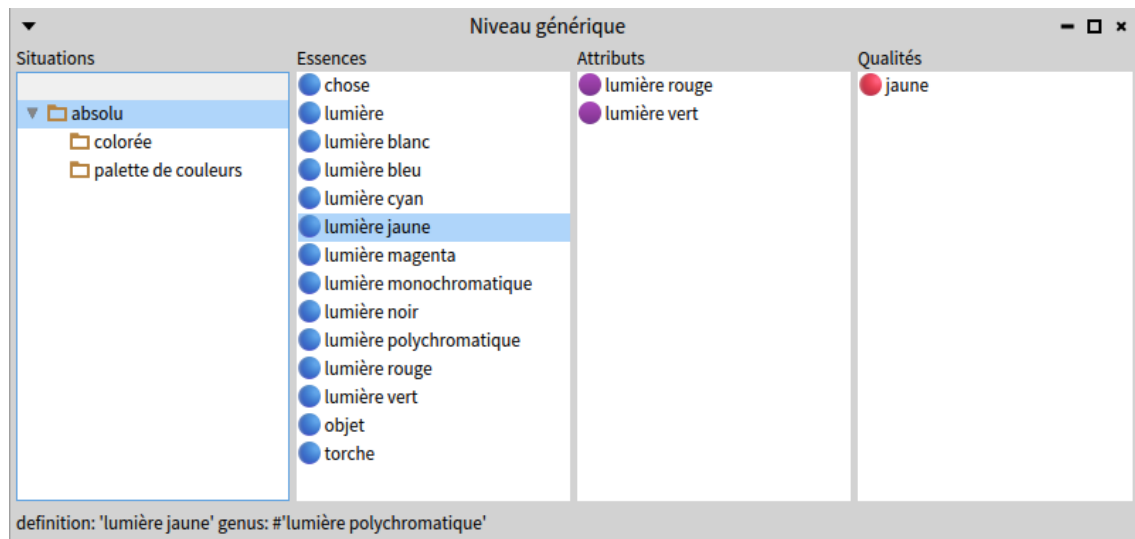
```

```

iceo definition: 'lumière vert' genus: (absolu get: 'lumière
monochromatique').
iceo definitionQualiteEssentielle: 'vert' pour: (absolu get: 'lumiè
re vert') effectivite: #permanente.
iceo definition: 'lumière rouge' genus: (absolu get: 'lumière
monochromatique').
iceo definitionQualiteEssentielle: 'rouge' pour: (absolu get: 'lumiè
ère rouge') effectivite: #permanente.
iceo definition: 'lumière cyan' genus: (absolu get: 'lumière
polychromatique').
iceo definitionQualiteEssentielle: 'cyan' pour: (absolu get: 'lumiè
re cyan') effectivite: #permanente.
(absolu get: 'lumière cyan') referenceEssence: (absolu get: 'lumiè
re bleu') cardinalite: 1.
(absolu get: 'lumière cyan') referenceEssence: (absolu get: 'lumiè
re vert') cardinalite: 1.
iceo definition: 'lumière magenta' genus: (absolu get: 'lumière
polychromatique').
iceo definitionQualiteEssentielle: 'magenta' pour: (absolu get: '
lumière magenta') effectivite: #permanente.
(absolu get: 'lumière magenta') referenceEssence: (absolu get: '
lumière bleu') cardinalite: 1.
(absolu get: 'lumière magenta') referenceEssence: (absolu get: '
lumière rouge') cardinalite: 1.
iceo definition: 'lumière jaune' genus: (absolu get: 'lumière
polychromatique').
iceo definitionQualiteEssentielle: 'jaune' pour: (absolu get: 'lumi
ère jaune') effectivite: #permanente.
(absolu get: 'lumière jaune') referenceEssence: (absolu get: 'lumiè
re vert') cardinalite: 1.
(absolu get: 'lumière jaune') referenceEssence: (absolu get: 'lumiè
re rouge') cardinalite: 1.
iceo definition: 'lumière blanc' genus: (absolu get: 'lumière
polychromatique').iceo definitionQualiteEssentielle: 'blanc'
pour: (absolu get: 'lumière blanc') effectivite: #permanente.
(absolu get: 'lumière blanc') referenceEssence: (absolu get: 'lumiè
re vert') cardinalite: 1.
(absolu get: 'lumière blanc') referenceEssence: (absolu get: 'lumiè
re rouge') cardinalite: 1.
(absolu get: 'lumière blanc') referenceEssence: (absolu get: 'lumiè
re bleu') cardinalite: 1.
iceo definition: 'lumière noir' genus: (absolu get: 'lumière
polychromatique').
iceo definitionQualiteEssentielle: 'noir' pour: (absolu get: 'lumiè
re noir') effectivite: #permanente.

```

L'ensemble de lumières ainsi défini est :



On voit que par exemple la lumière jaune possède la qualité essentielle d'être jaune, et qu'elle est composée des lumières rouge et vert.

### Couleur des objets

La couleur d'un objet est celle de la lumière qu'il reflète ou diffuse lorsqu'il est éclairé.

un objet éclairé avec une lumière peut être coloré comme il peut être mouillé avec un liquide.

Les capacités de diffusion de la lumière d'un objet sont liées à sa composition physique; elles ne dépendent pas de l'éclairage mais déterminent sa couleur une fois éclairé. Elles sont bien sûr les mêmes en l'absence de lumière (dans le noir).

En peinture, la composition des couleurs est basée sur les capacités de diffusion de la lumière par la peinture. La composition est soustractive. Les trois couleurs fondamentales en peinture sont cyan, magenta et jaune. La peinture verte est obtenue par mélange de pigments jaune et cyan :



Par convention, la couleur d'un objet est celle qu'il possède éclairé par de la lumière blanche. C'est sa couleur prise conventionnellement par défaut, que nous qualifierons de couleur "objective". Avec cette convention, éclairé par de la lumière rouge, un objet jaune paraîtra rouge (car la couleur objective jaune signifie que l'objet diffuse les lumières rouge et vert).

Quel que soit l'éclairage, un objet noir apparaîtra toujours noir, car il absorbe toutes les lumières.

Nous qualifierons de "couleur apparente" la couleur que prend un objet avec un éclairage particulier.

Autrement dit, c'est le couple (couleur objective, éclairage) qui détermine la couleur apparente de l'objet. Changer l'éclairage entraîne un changement de couleur apparente

des objets éclairés. En l'absence d'éclairage, c'est-à-dire dans le noir, un objet paraît noir.

L'éclairage est le contexte où la couleur d'un objet devient perceptible.

Voici la définition d'un objet de couleur jaune :

```
iceo definition: #objet.objet peutEtre: ((absolu getSituation: #
    colorée) get: #couleur).
iceo soit: 'un objet' essence: objet.
(monde get: 'un objet') affecteEtat: ((absolu getSituation: '
    palette de couleurs') getElement: #jaune) dansSituation: (monde
    get: 'une palette de couleurs').
```

Et voici la définition d'une torche émettant une lumière magenta :

```
iceo definition: #torche.torche referenceEssence: lumière.
iceo soit: 'une torche' essence: torche.
(monde get: 'une torche') attributionEtre: 'une lumière' essence: (
    absolu get: 'lumière magenta').
```

Le code Smalltalk suivant définit une méthode de l'essence objet qui infère la couleur apparente d'un objet éclairé par une torche, en faisant l'intersection de l'ensemble de lumières émises par la torche avec l'ensemble des lumières reflétées par l'objet :

```
objet compile: 'couleurApparenteEclairePar: uneTorche
| l c s t |
s := (absolu getElements: lumière) select: [:each |
    (each getGenus == (absolu get: "lumière monochromatique")) or: [each getGenus == (absolu get: "lumière polychromatique")]].
t := s detect: [:each |
    l := each.
    l getGenus == (absolu get: "lumière monochromatique")]

ifTrue: [c := OrderedCollection with: l]
ifFalse: [c := l getEssencesAttributs sort: [:a :b | a getNom < b getNom]] .
c = ((uneTorche lumièresEmises intersection: self lumièresReflétées) sort: [:a :b | a getNom < b getNom]) .
^t getQualites at: 1 '.
```

Cette méthode utilise deux autres méthodes qui déterminent respectivement les lumières reflétées par un objet et les lumières émises par une torche :

objet compile: 'lumièresReflétées

| | |

l := (absolu get: "lumière ", (self getEtatEssence: ((absolu getSituation: #colorée) get: #couleur)) getNom).

l getGenus == (absolu get: "lumière monochromatique") ifTrue: [^ OrderedCollection with: l ]

ifFalse: [ ^ l getEssencesAttributs sort: [:a :b | a getNom < b getNom] ] '.

torche compile: 'lumièresEmises

| | |

l := (self getEtreAttribut: #lumière) getEssence.

l getGenus == (absolu get: "lumière monochromatique") ifTrue: [^ OrderedCollection with: l ]

ifFalse: [ ^ l getEssencesAttributs sort: [:a :b | a getNom < b getNom] ] '.

Ainsi par exemple, pour un objet de couleur jaune éclairé par de la lumière magenta, l'expression :

```
(monde get: 'un objet') couleurApparenteEclairePar: (monde get: 'unetorche')
dans: monde
```

retourne : rouge

Pour un objet de couleur jaune éclairé par de la lumière bleue, la couleur apparente serait le noir, ...



## Exemple 22 : les tours de Hanoï revisitées

Au XIX<sup>e</sup> siècle, le mathématicien français Edouard Lucas a inventé le jeu des tours de Hanoï, une simple récréation mathématique qui s'est révélée au fil des années une mine de réflexions.

Le jeu des tours de Hanoï est constitué de trois piquets placés verticalement, et de  $n$  disques de taille décroissante. Les disques sont placés initialement par taille décroissante sur le premier piquet. Le but du jeu consiste à déplacer les disques jusqu'à parvenir à la situation finale dans laquelle tous les disques se retrouvent autour d'un autre piquet par ordre de taille décroissante. Les disques se déplacent en suivant deux règles :

- on ne déplace qu'un seul disque à la fois
- un disque ne peut jamais être placé sur un disque plus petit.

Ce problème est un cas d'école d'algorithme dit récursif. Pour  $n$  disques, le nombre minimum de déplacements est  $2^n - 1$

La solution que nous proposons ici est inspirée de celle proposée par Ted Kaehler et Dave Patterson dans le livre "A TASTE OF SMALLTALK".

Dans leur solution, les disques sont des objets qui se déplacent en fonction des messages déterminés par un chef d'orchestre. Aux deux règles du jeu est ajoutée une troisième (qui semble logique pour avoir un minimum de déplacements) interdisant qu'un disque ne revienne en arrière. Leur algorithme exige que des disques fictifs (mock disks) de taille plus grande que celle des autres soient placés sur les piquets vides.

Notre solution diffère sur différents points :

- les disques sont des acteurs (implantés sous forme de processes) qui jouent une course de relais.
- nous avons défini les manières d'être relatives "sous" et "sur" pour les disques (ce qui évite d'avoir recours à des "mock disks" fictifs).

L'algorithme utilisé est également original.

Pour l'exemple, le nombre de disques a été fixé à 9.

Ce nombre peut être bien sûr être changé. Noter que si vous fixez ce nombre à 64, le nombre minimum de déplacements sera de  $2^{64} - 1$ , soit 18 446 744 073 709 551 615. Il faudra être patient !

Pour que le déplacement des disques soit visible, un délai paramétrable (qui peut être nul) de 10 ms a été imposé entre deux déplacements.

```
iceo definition: #jeu.iceo definitionSituation: #piquet.  
jeu referenceEssence: piquet cardinalite: 3.  
iceo definitionAttribut: #disque de: jeu cardinalite: 9.  
Smalltalk at: #delay put: 10. "ms"
```

Les piquets, au nombre de trois, sont les situations entre lesquelles évoluent les disques.

Au départ, les disques sont empilés sur le premier piquet en tenant compte de leur taille.

Chaque disque possède un comportement qui est une manière d'être essentielle et permanente.

```
iceo definitionQualiteEssentielle: #comportement pour: (jeu
  getEssenceAttribut: #disque) effectivite: #permanente.
```

Au comportement de chaque disque est associé un process Smalltalk et un sémaphore<sup>1</sup> :

```
" définition du rôle de chaque joueur "
(monde get: #Hanoi) getEtres do: [:each |
  | started |
  started := false.
  (each getEtat: #comportement) setSemaphore: Semaphore new.
  (each getEtat: #comportement) setProcess: ([ :h :d |
    | s |
    s isNil ifTrue: [ s := false ].
    [ true ] whileTrue: [
      s ifFalse: [
        (d getNom , ' is ready') crTrace.
        s := true.
        (d getEtat: #comportement) getSemaphore wait ].
      d jyVais.
      "qui peut maintenant prendre le relais ? "
      h getEtres
        detect: [:disque | disque peutSeDeplacer ]
        ifFound: [:x | (x getEtat: #comportement) getSemaphore signal ].
      (d getEtat: #comportement) getSemaphore wait ] ]
    newProcessWith: (Array with: (monde get: #Hanoi) with: each)) ].
```

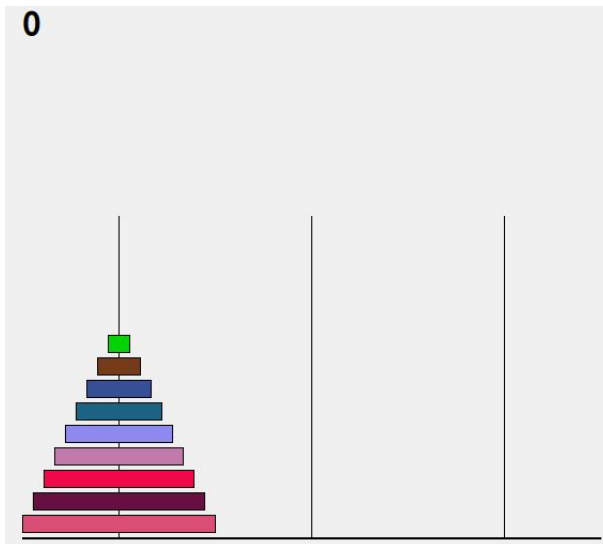
Au départ, les joueurs sont placés dans un état d'attente par "allezAuxStartingBlocks".

Pour initialiser le jeu, sélectionnez et exécutez l'ensemble des instructions de l'exemple (où une bonne partie du code est associée au graphisme).

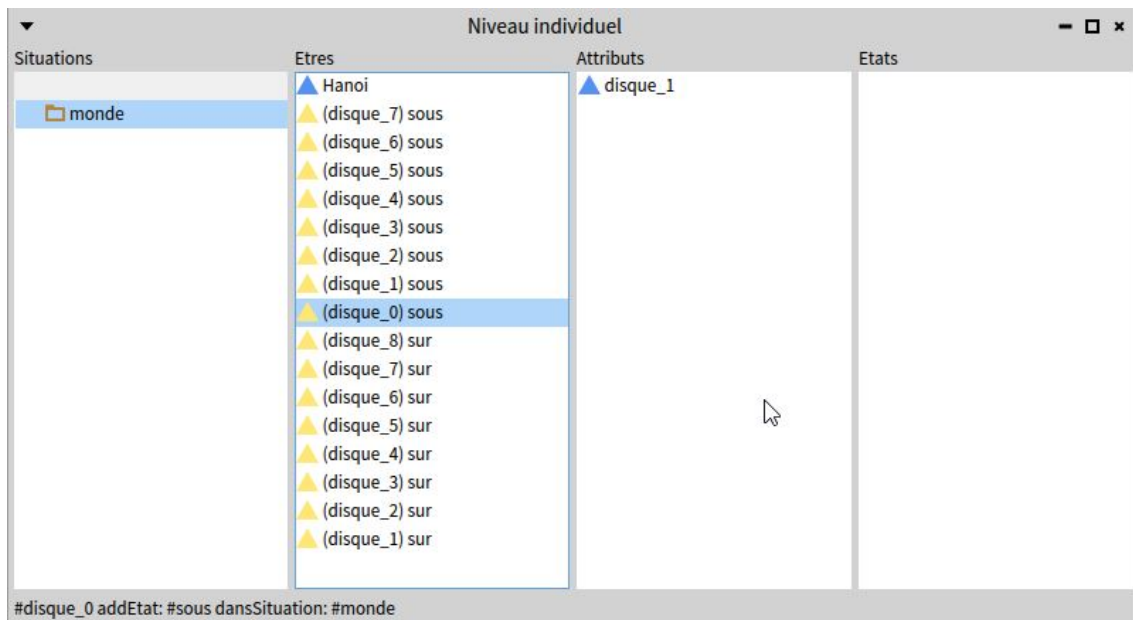
L'image suivante s'affiche (la couleur des disques est déterminée de manière aléatoire) :

---

<sup>1</sup>Les notions de process et de sémaphore en Smalltalk sont décrits dans le livre "Concurrent Programming in Pharo" de Stéphane Ducasse et Guillermo Polito".



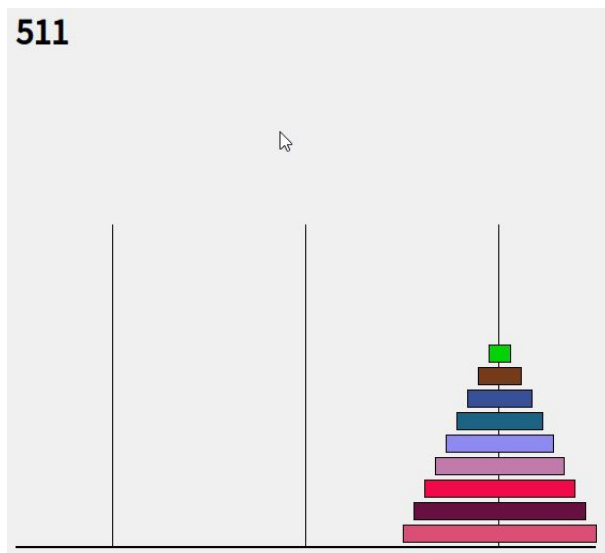
Il peut être intéressant à ce stade d'ouvrir un SiBrowser :



On peut vérifier qu'au départ les disques 0 à 7 sont dans l'état "sous" et que les disques 1 à 8 sont dans l'état "sur".

Un click avec le bouton gauche de la souris sur l'un des disques envoie un signal au plus petit pour lui dire de commencer la course.

Vous voyez les disques se déplacer dans une course de relais qui se termine lorsque les disques ont retrouvé leur configuration initiale, sur un autre piquet.



Le nombre affiché en haut à gauche de l'image correspond au nombre de déplacements effectués par les disques. Le nombre 511 correspond à la valeur  $2^9-1$  qui est la valeur théorique minimale pour ce jeu avec neuf disques.

Pour clore le jeu, il suffit d'un click avec le bouton droit de la souris sur l'un des disques (ou le nombre de déplacements).

# Sur l'implantation d'ICEO en Pharo

Il y a peu de choses à dire sur cette implantation, tant ICEO s'intègre naturellement dans le langage Smalltalk.

Aucune méthode de l'environnement de base de Pharo n'a été modifiée, à part la méthode "evaluate: aString onCompileError: compileErrorBlock onError: errorBlock" de SpCodePresenter, pour éviter que le compilateur trop pressé n'évalue une expression avant d'avoir terminé d'évaluer la précédente ! (ce qui permet d'évaluer toutes les expressions de chaque exemple en bloc).

Cette méthode attrape les exceptions de classe Oups.

La méthode "validateClassName" de ShiftClassBuilder a également été modifiée pour autoriser que le nom des classes ne commencent pas par une majuscule. Pour permettre l'homonymie des essences, nous avons mis à profit la possibilité de créer des classes de type "newAnonymousSubclass".

La classe Essence a comme sous-classes directes les essences "chose" (racine de la hiérarchie des essences dans ICEO) et "absolu" qui est la racine des situations génériques.

Les attributs des essences sont des instances de la classe Association, ce qui permet d'associer à chaque essence attribut une cardinalité pour sa composition.

Pour la composition des essences qui sont des manières d'être et des situations génériques, celles-ci sont traitées comme des essences (avec leur differentia), à la différence près que leurs attributs n'ont pas de cardinalité.

La classe ICEO correspond à l'interprète d'ICEO, utilisée pour la définition des situations, des essences et de leurs qualité et l'instanciation des êtres.

La classe Etre est superclasse de la classe Essence. Les attributs et les méthodes de classe de la classe Etre sont les mêmes que ceux des instances de la classe Essence, ce qui permet de voir toute essence comme un être instance de sa propre essence (sa méta-essence).

Par défaut, la méta-essence d'une essence est l'essence chose et l'essence chose est son propre genus et sa propre méta-essence.