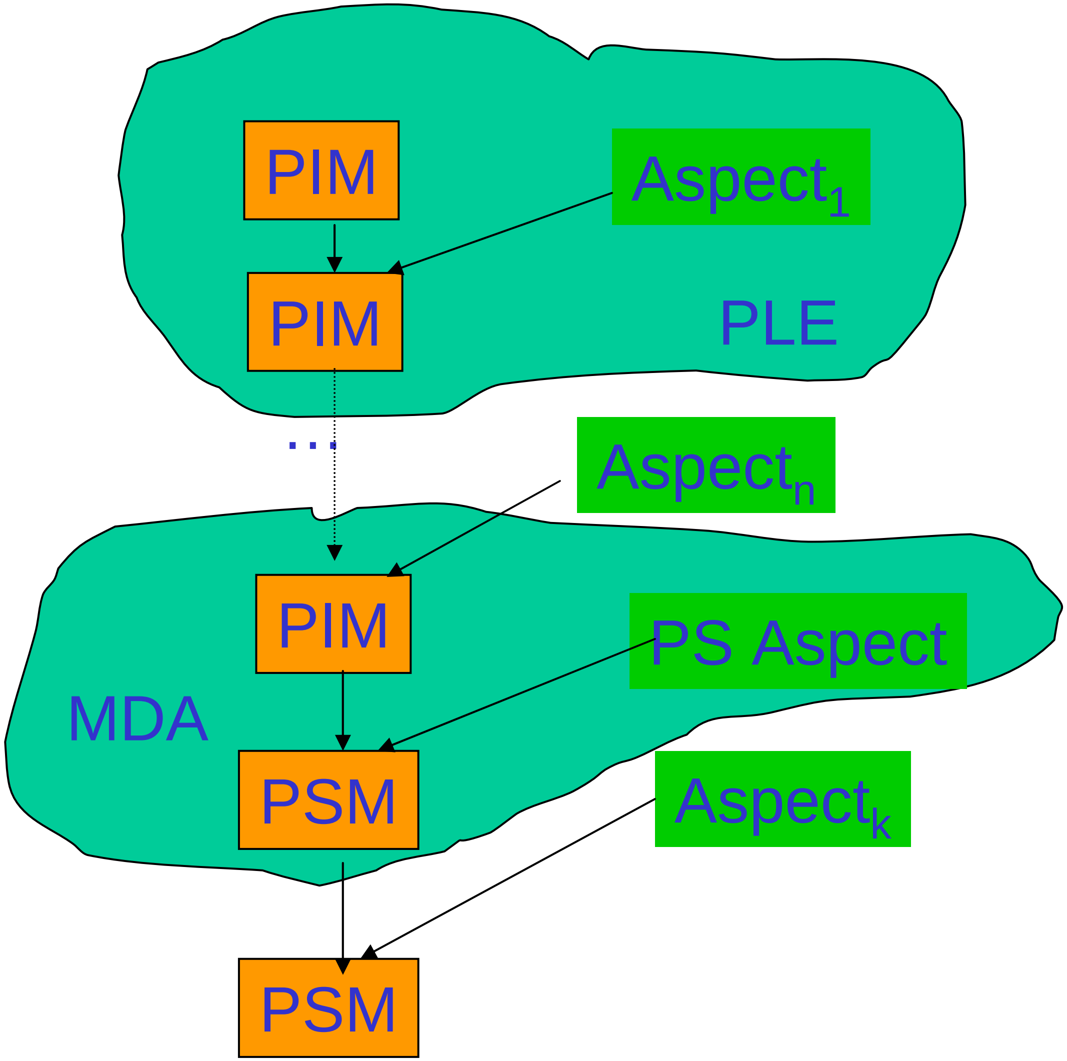
Product Line Engineering :
Product Derivation in an MDA framework

Tewfik Ziadi
Jean-Marc Jézéquel
Frédéric Fondement

WP3 – Task 3.3

MDA and PLE : a Vision



Aspect weaving is The unifying paradigm !

MDA is mainly useful in a PLE context

Product derivation in a Product Line: Executive Summary

- The analysis model has identified the *variants* between products (including Platform specificities)
  - Reified as language-level classes (inheritance, …)
  - Decorated with OCL meta-level constraints
- Systematic use of the *Abstract Factory* pattern
  - To specify a product among the family
- Model Transformations (at the meta-model level) to automatically derive a product
  - Using OCL2

The 3 Dimensions of Software Configuration Management : [Estublier et al. 95]

The Variant dimension

- Handle environmental differences

The revision dimension

- Evolution over time

The concurrent activities dimension

- Many developers are authorized to modify the same configuration item

Even with the help of sophisticated tools, the complexity might be daunting

Try to simplify it by reifying the variants of an OO system

Traditional Approaches

Patch the executable

- Device drivers
  - Source level, link time, boot time, on demand at runtime
- Static configuration table
- Conditional Compilation / Runtime Tests

```
If (language == french) {
  #ifdef MSW
               io_puts(0, ‘‘Bonjour’’, 7);
  #elseif TEXT
               printf(‘‘Bonjour\n’’);
  #endif
} else {
  #ifdef MSW
               io_puts(0, ‘‘Hello’’, 5);
  #elseif TEXT
               printf(‘‘Hello\n’’);
  #endif
```

- Static and Dynamic configuration information intermingled
- Hard to change your mind on what should be static or dynamic…

Basic Idea

Abstract the Intent

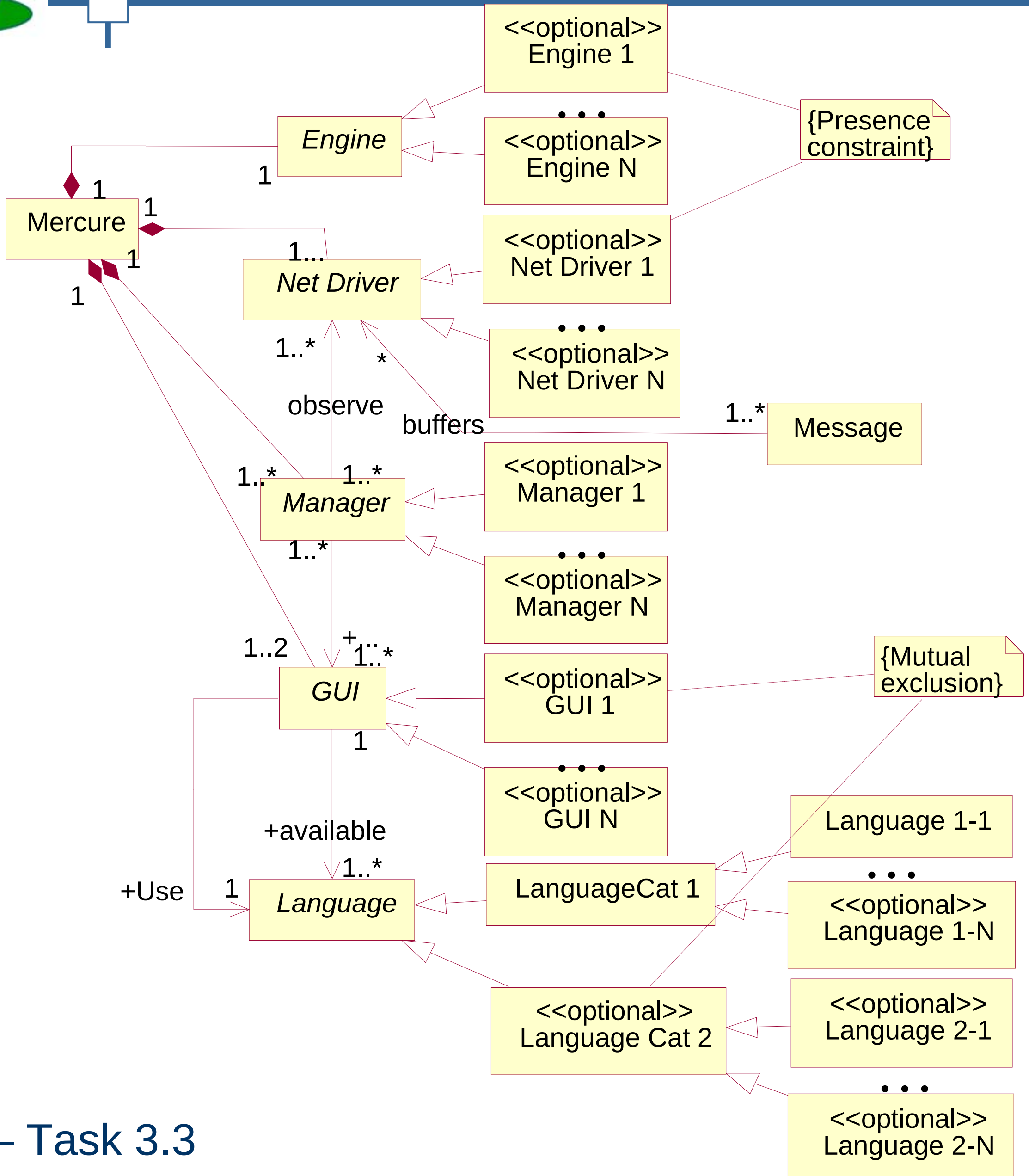
- Io.write_line(language.hello)

Rely on Dynamic Binding for the Details

Uncouple the variations from the selection process

- Automatically derive a product using OCL2 meta-model transformation

Case Study: The Mercure Product Line



43,980,465,111,040

possible variants

Exemples of Constraints at the PLA Level

Inheritance constraint

- Optional classes in Product Line Architecture can be omitted in certain products so a non-optional class cannot inherit from an optional one.
- OCL expression (at the M2 level):
  - **context** Generalization
    inv self.parent.isStereotyped(“optional”) **implies**
    self.child.isStereotyped(“optional”)

Dependency constraint

- Idem
  - **context** Dependency
    inv self.supplier->**exists**(S:ModelElement | .isStereotyped(“optional”))
    implies self.client->**forAll**(C:ModelElement |
    C.isStereotyped(“optional”))

Constraints specific to a given PLA

Presence constraint.

- To express in a specific SPLA that the presence of the optional class C1 requires the presence of C2, we add the following OCL meta-model constraint.

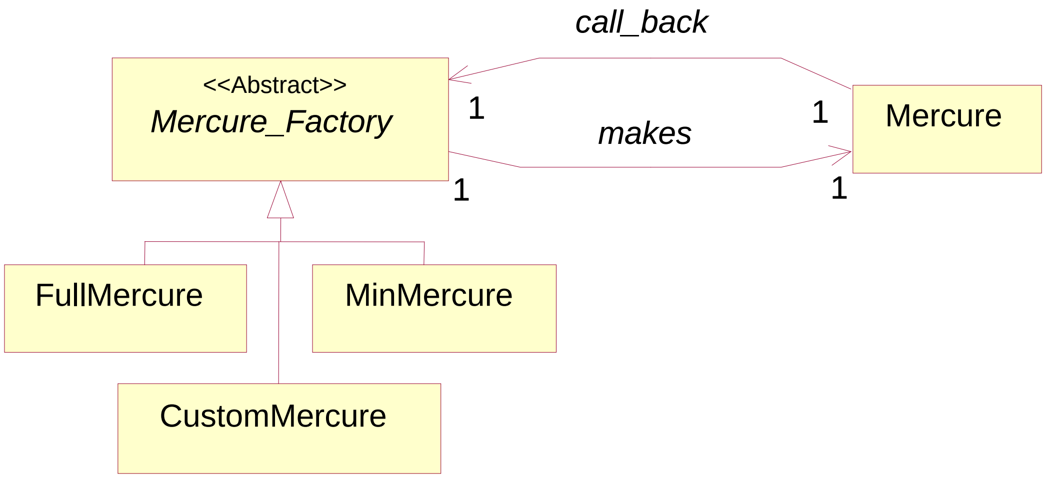
  -
    ```
    context Model
       inv presenceClass(‘C1’) implies presenceClass(‘C2’)
    ```

Mutual Exclusion constraint.

- To express in a specific SPLA that two optional classes cannot be present in the same Product, we add the following OCL meta-model constraint.

  -
    ```
    context Model
       inv    (presenceClass(‘C1’) implies not presenceClass(‘C2’))
       and    (presenceClass(‘C2’) implies not presenceClass(‘C1’))
    ```

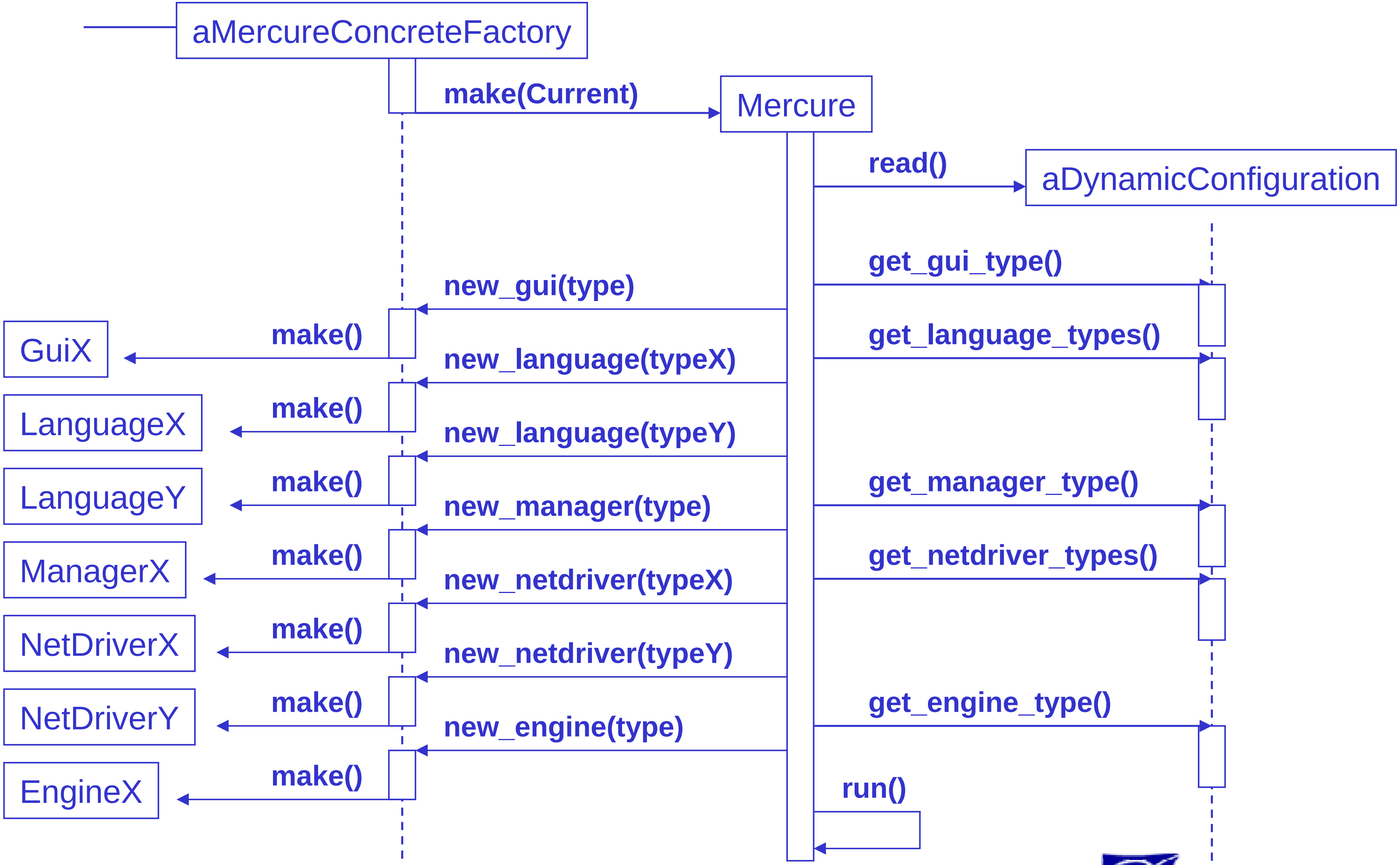
Reifying the Variants



CustomMercure

<<GUI1>> <<GUI2>>new_gui() : GUI
<<<Language2-1>> new_language() : Language
<<Manager1>> new_network_manager() : Manager
<<NetDriver1>> <<NetDriver2>>new_netdriver() : Net Driver
<<Engine1>> new_engine() : Engine

Dynamic Configuration



Model Transformation

By limiting the range of variants available from a given Concrete Factory:

- Generate specialized code for the product
  - When only one *living* class for an abstract varying part:
    - Direct use of the relevant Concrete Class =>
    - Dynamic binding replaced with direct call (and even in lining)
  - When more than one *living* class
    - Dynamic binding (or replaced by *if then … else)*

Implemented in SCM context using compilation – GNU SmallEiffel

- “Reifying variants in configuration management” J.ML. Jézéquel. *ACM Transactions on Software Engineering and Methodology*, July 1999.

Using OCL2 & UMLAUT framework

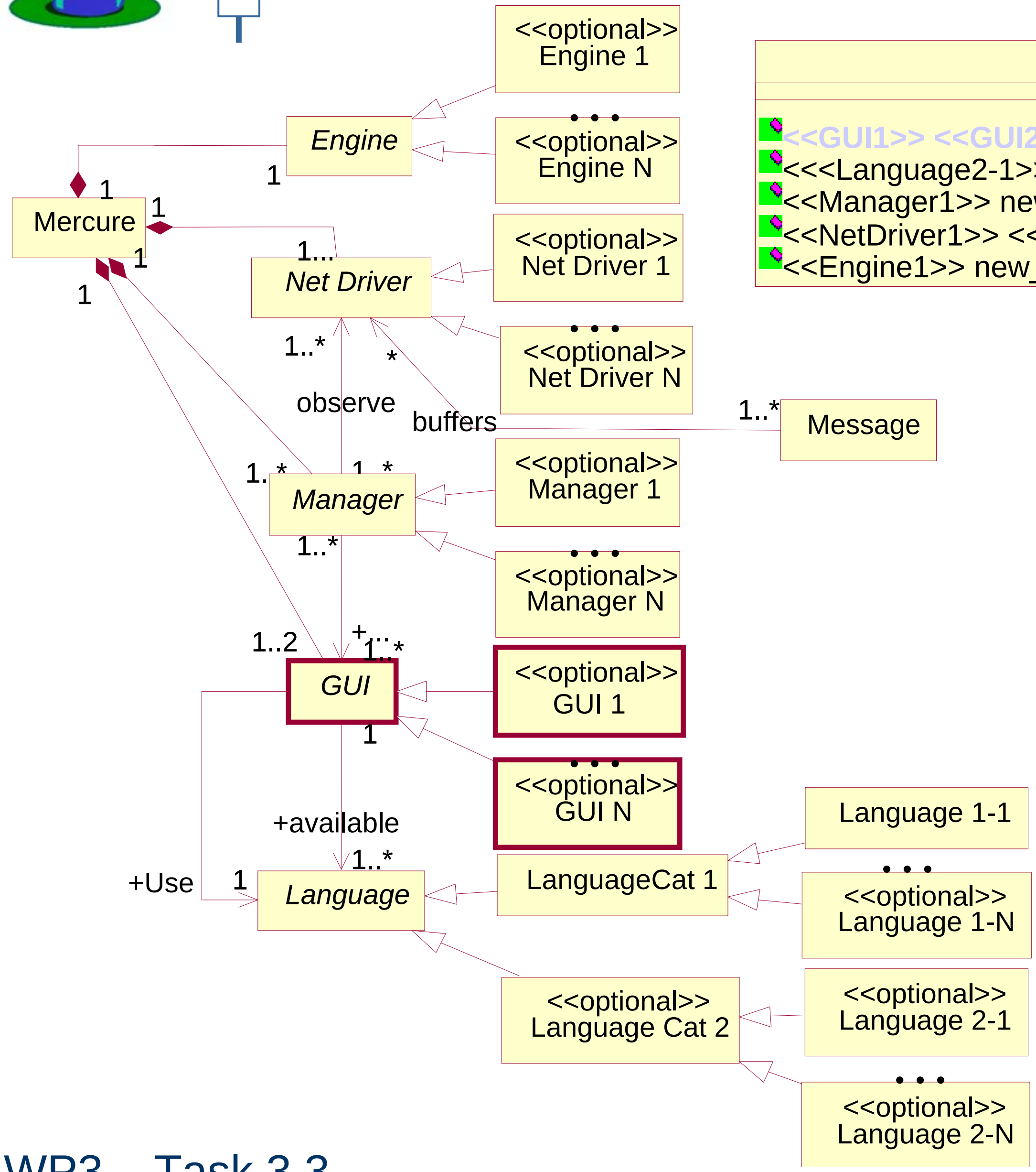
Using OCL2

```
forAll op in Operation.allInstances() {

  -- The returned type of the operation determines the used variants
  Class opReturnType :=
    (op.parameter->select(p:Parameter|p.kind = #return)).type

  if opReturnType.isAbstract
  then
    -- For multiple choice, we use stereotypes to specify the
    -- selected variants
    forAll st in op.stereotype {
      selectVariant(st.name)
    }
  else
    -- Here, we directly get the selected variant
    selectVariant(opReturnType.name)
  endif
}
```

Class Diagram Handling



Engine

<<optional>> Engine 1

• • •

<<optional>> Engine N

Mercure

Net Driver

<<optional>> Net Driver 1

• • •

<<optional>> Net Driver N

Message

Manager

<<optional>> Manager 1

• • •

<<optional>> Manager N

GUI

<<optional>> GUI 1

• • •

<<optional>> GUI N

Language

LanguageCat 1

Language 1-1

• • •

<<optional>> Language 1-N

<<optional>> Language Cat 2

<<optional>> Language 2-1

• • •

<<optional>> Language 2-N

1

1

1

1

1

1...

1..*

*

observe

buffers

1..*

1..*

1..*

1..*

1..2

1..*

1

+available

1..*

+Use

1

CustomMercure

🟩<<GUI1>> <<GUI2>>new_gui() : GUI

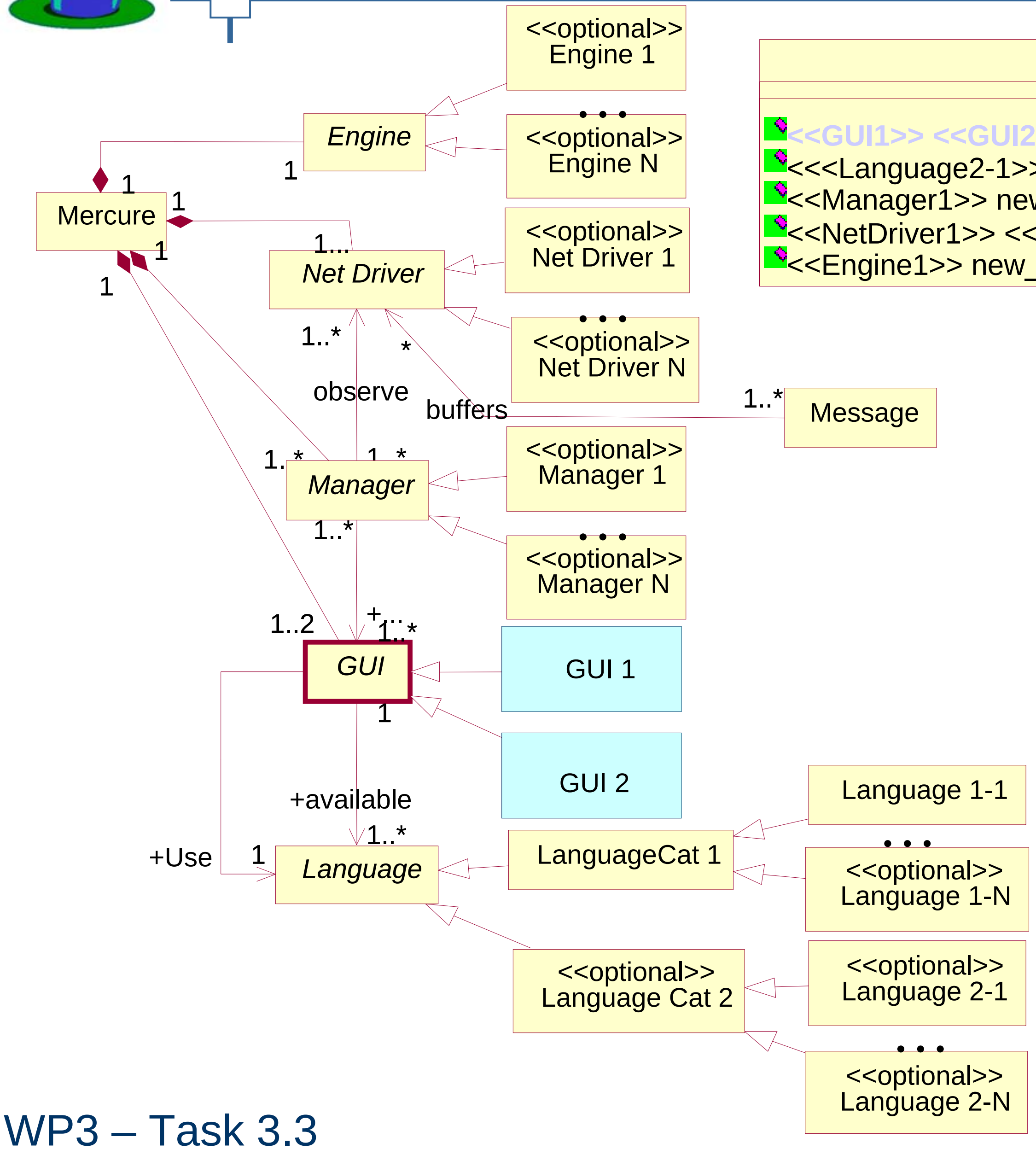
🟩<<<Language2-1>> new_language() : Language

🟩<<Manager1>> new_network_manager() : Manager

🟩<<NetDriver1>> <<NetDriver2>>new_netdriver() : Net Driver

🟩<<Engine1>> new_engine() : Engine

Class Diagram Handling



Mercure

Engine

<<optional>>
Engine 1

• • •

<<optional>>
Engine N

Net Driver

<<optional>>
Net Driver 1

• • •

<<optional>>
Net Driver N

Message

Manager

<<optional>>
Manager 1

• • •

<<optional>>
Manager N

GUI

GUI 1

GUI 2

Language

LanguageCat 1

Language 1-1

• • •

<<optional>>
Language 1-N

<<optional>>
Language Cat 2

<<optional>>
Language 2-1

• • •

<<optional>>
Language 2-N

1

1

1

1

1

1...

1..*

*

observe

buffers

1..*

1..*

1..*

1..*

1..2

+1..*

1

+available

1..*

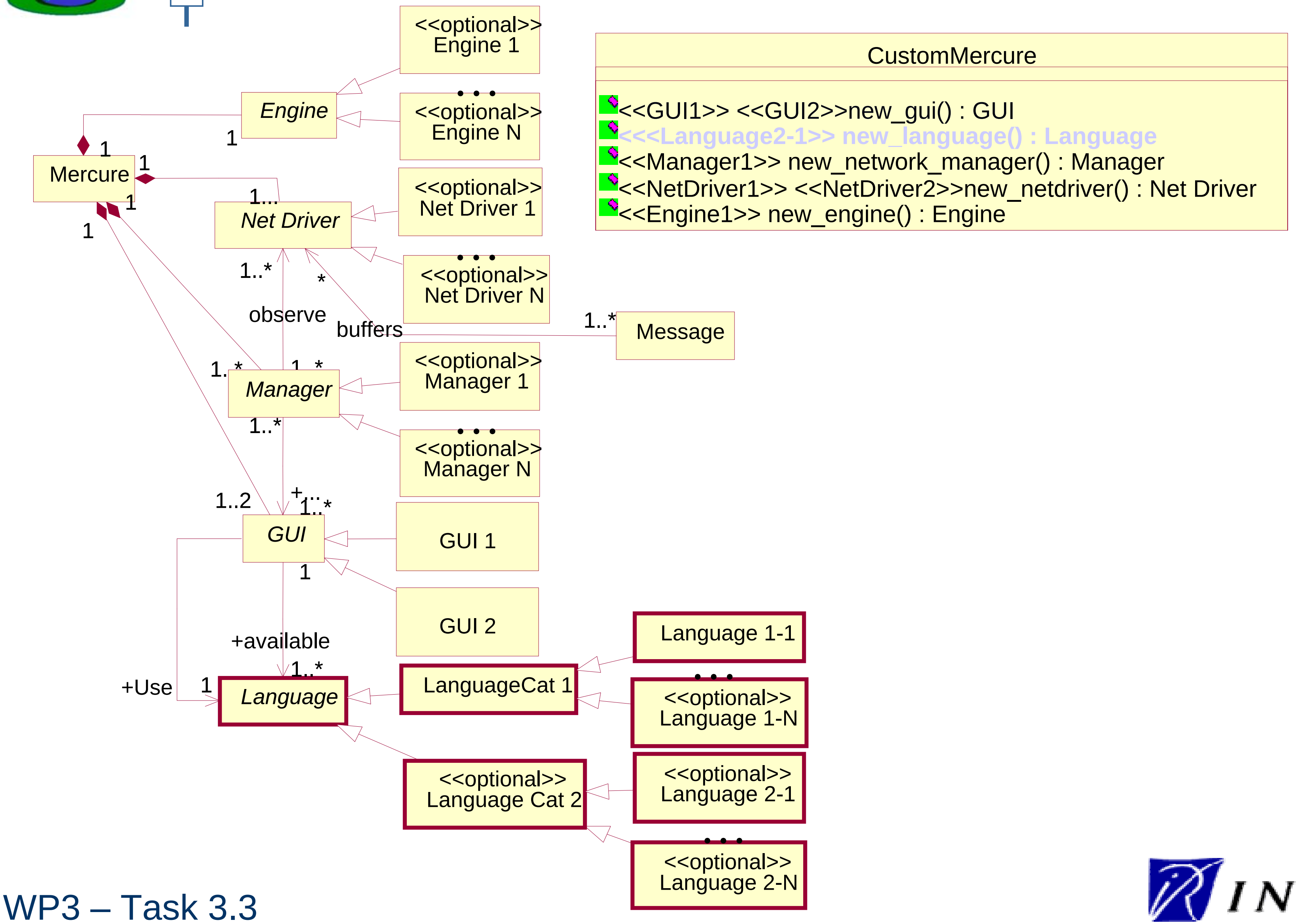
+Use

1

CustomMercure

<<GUI1>> <<GUI2>>new_gui() : GUI
<<<Language2-1>> new_language() : Language
<<Manager1>> new_network_manager() : Manager
<<NetDriver1>> <<NetDriver2>>new_netdriver() : Net Driver
<<Engine1>> new_engine() : Engine

Class Diagram Handling



CustomMercure

<<GUI1>> <<GUI2>>new_gui() : GUI

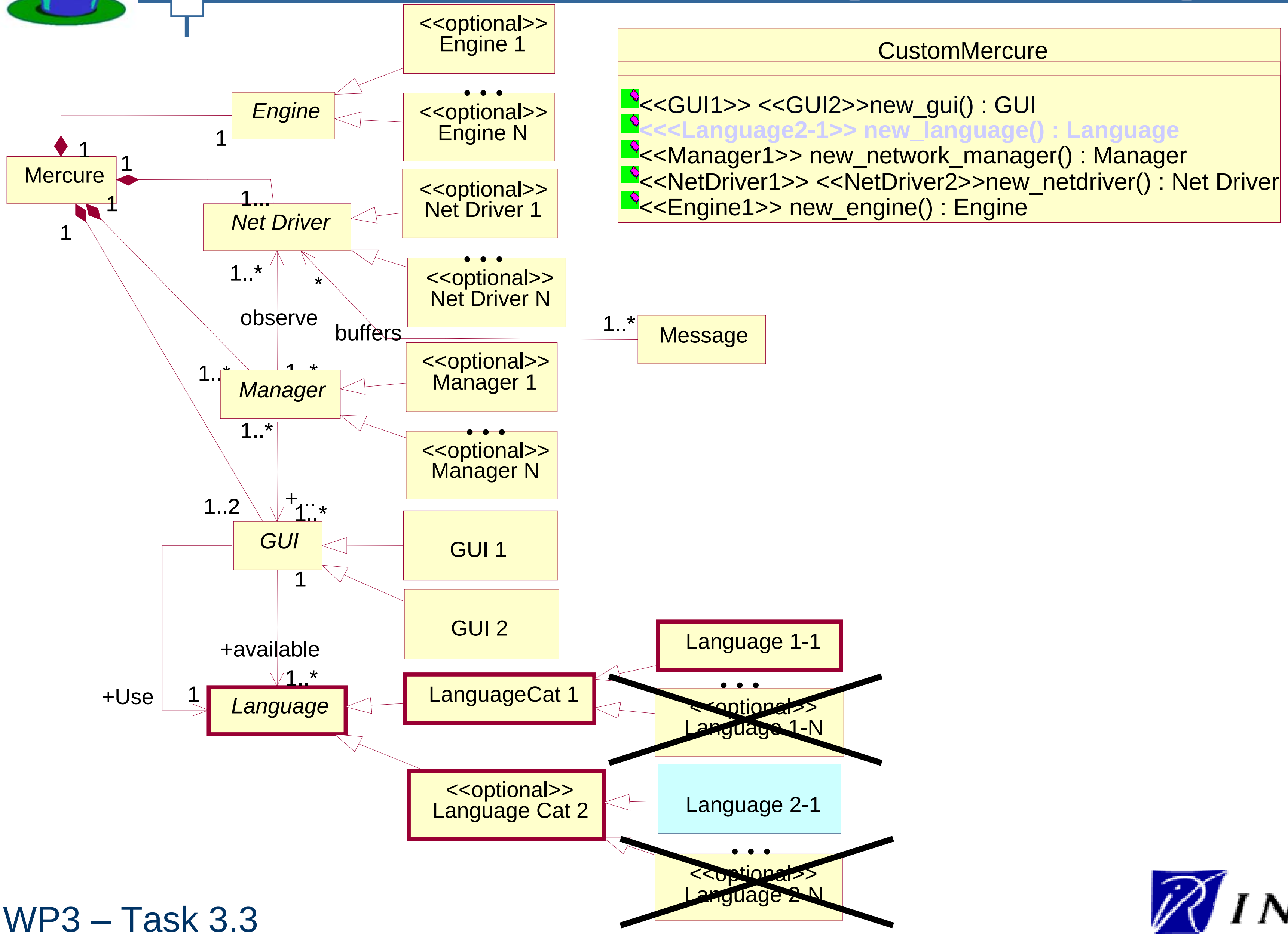
<<<Language2-1>> new_language() : Language

<<Manager1>> new_network_manager() : Manager

<<NetDriver1>> <<NetDriver2>>new_netdriver() : Net Driver

<<Engine1>> new_engine() : Engine

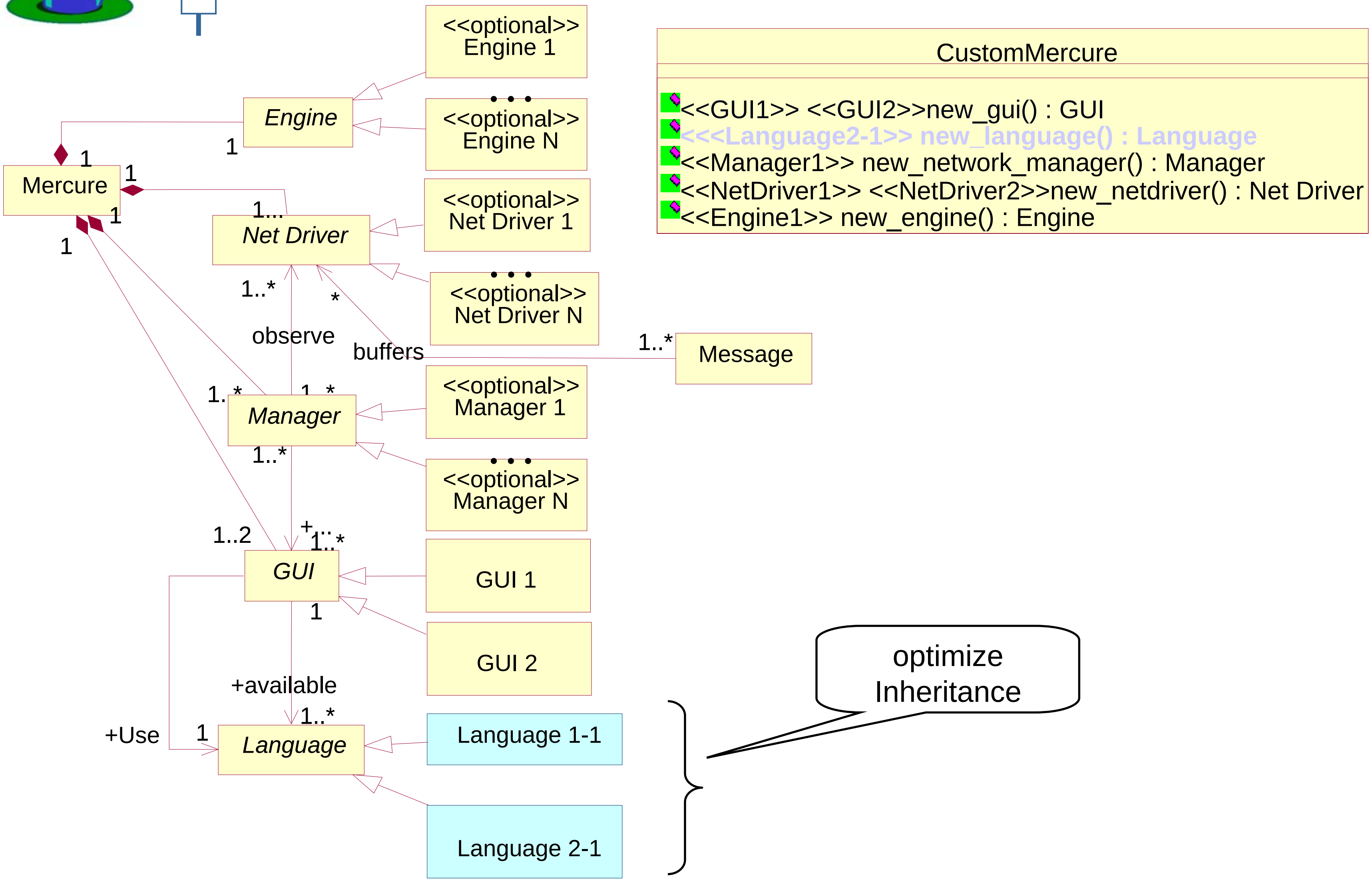
Class Diagram Handling



CustomMercure

🟩<<GUI1>> <<GUI2>>new_gui() : GUI
🟩<<<Language2-1>> new_language() : Language
🟩<<Manager1>> new_network_manager() : Manager
🟩<<NetDriver1>> <<NetDriver2>>new_netdriver() : Net Driver
🟩<<Engine1>> new_engine() : Engine

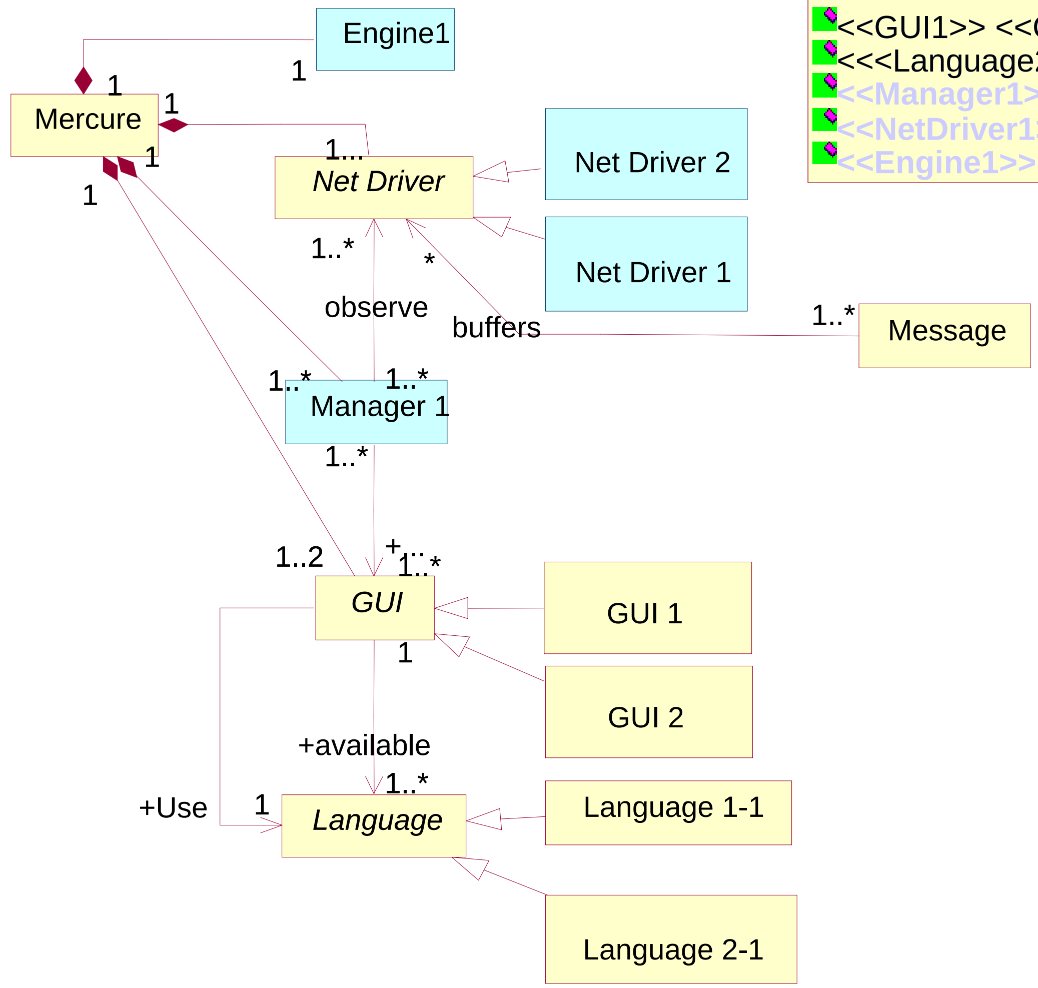
Class Diagram Handling



CustomMercure

<<GUI1>> <<GUI2>>new_gui() : GUI
<<<Language2-1>> new_language() : Language
<<Manager1>> new_network_manager() : Manager
<<NetDriver1>> <<NetDriver2>>new_netdriver() : Net Driver
<<Engine1>> new_engine() : Engine

Class Diagram Handling

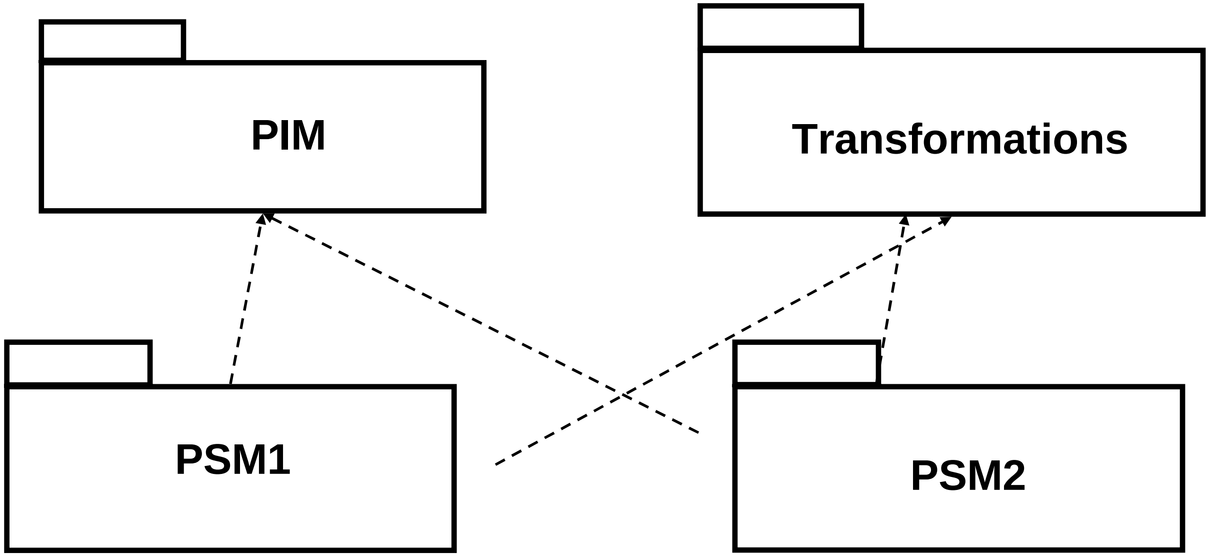


CustomMercure

🟩<<GUI1>> <<GUI2>>new_gui() : GUI
🟩<<<Language2-1>> new_language() : Language
🟩<<Manager1>> new_network_manager() : Manager
🟩<<NetDriver1>> <<NetDriver2>>new_netdriver() : Net Driver
🟩<<Engine1>> new_engine() : Engine

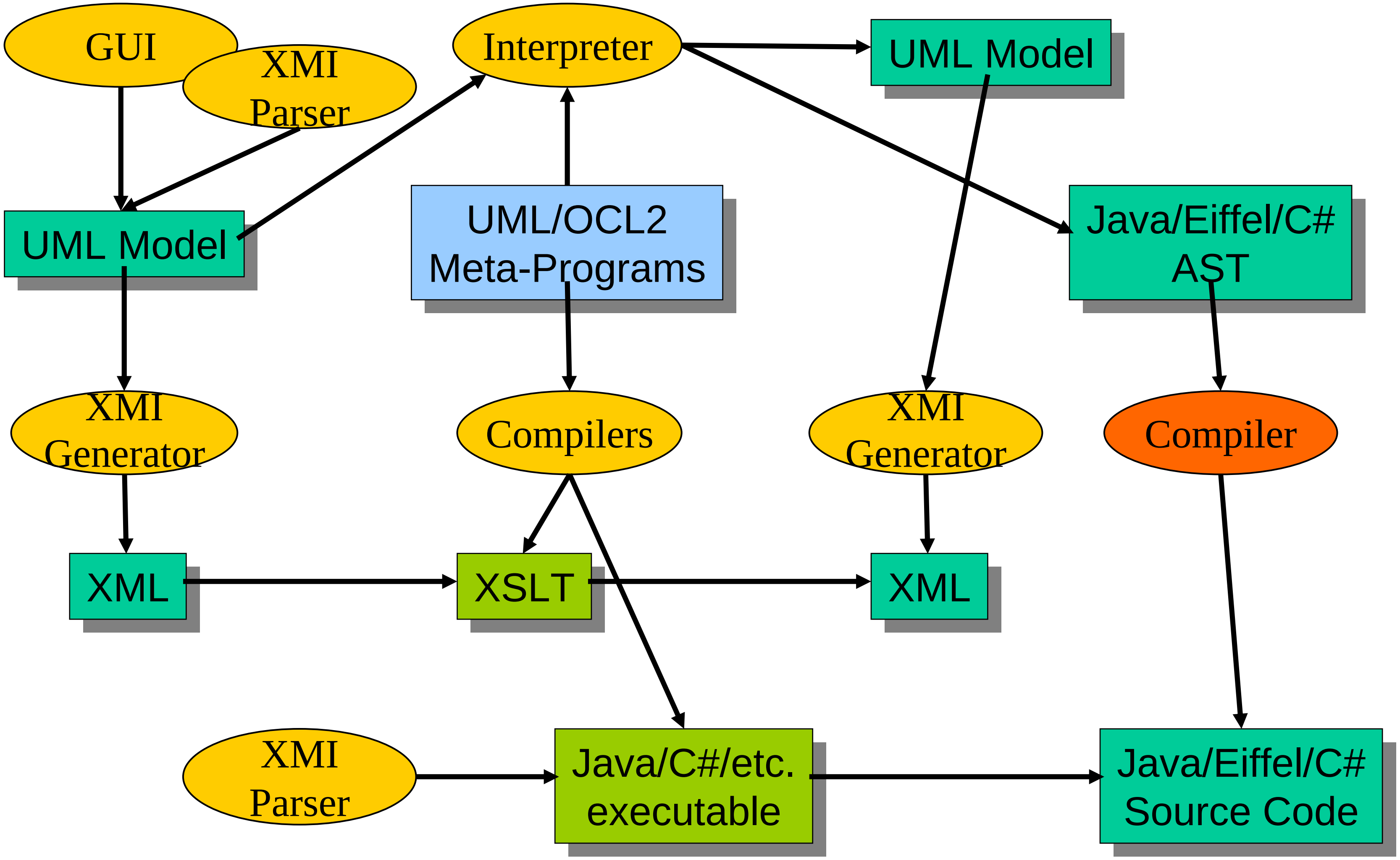
Class Diagram Handling: Ongoing work

Model of PIM and Model of Transformation side by side on the CASE tool



Use a meta-level OCL2 interpreter/compiler

Our UMLAUT New Generation



Conclusion

- Method to uncouple the variations (reified as language-level objects) from the selection process
  - Based on the use of Creational Design Patterns
    - Abstract Factory
- All static configuration issues kept encapsulated in the Concrete Factory
- Model transformations with OCL2 makes it attractive
  - Experimented with GNU SmallEiffel compiler
  - Generalize the idea to UML using OCL2
  - “Constraint preserving” transformations

Derivation algorithm

- Pseudo-code

Derivation algorithm

```
Input:    PL_model: Model
          aConcreteFactory: Class
Output :  Product_model: Model

--Optional elements selection
Initiate selectedVariantsList to empty;
for each factory method in
    aConcreteFactory do
    initiate definedVariantsList to
        significant stereotypes of the factory;
    if definiedVariantsList is empty
        then selectedVariantsList.add(
            all sub classes of the returned type);
    else
        selectedVariantsList.add(definedVariantsList) ;
    endif
done
```

```
-- Model specialization
for each optional class C in PL_model do
    if (the class name of C not in
selectedVariantsList) and ( names of all sub
classes of C not in selectedVariantsList)
    then
        delete the class C from the PL_model;
    endif
done
-- Model optimization
delete all other factories;
optimize inheritance;
Product_model := PL_model;
```