|  |
|---|
| REuse and Migration of legacy applications to Interoperable Cloud Services |
| **REMICS** |
| Small or Medium-scale Focused Research Project (STREP) |
| Project No. 257793 |

## Deliverable D3.2

# REMICS Recover Toolkit, Preliminary release

**Work Package 3**

Leading partner: Netfective

Author(s): Franck Barbier, Gaëtan Deltombe, Alexis Henry

Dissemination level: PU

**Delivery Date: August 31st, 2011**

**Final Version: 1.0**

## Versioning and contribution history

| Version | Description | Contributors |
|---|---|---|
| 0.1 | Initial version | Franck Barbier |
| 0.2 | Reviewed version by Brice M. and Andrey S. | Franck Barbier |
| 1.0 | Final version | |

# Executive Summary

This document intends to comment on the D3.2 deliverable of REMICS that consists in **a product** (see DoW). More precisely, D3.2 is a set of *Eclipse Modeling Framework* metamodels: *Knowledge Discovery Metamodel (KDM)*, *Abstract Syntax Tree Metamodel (ASTM)* and *Extension of Knowledge Discovery Metamodel* (EKDM) that intend to be used in model-driven modernization tools. In REMICS, BLU AGE® from Netfective, a proprietary tool, is used as a metamodel processor to cope with D3.2 metamodels:

- KDM: implementation is available from *kdmanalytics.com*;
- ASTM: **ASTM.ecore** (see also D3.1);
- EKDM: **EKDM.ecore** (see also D.1).

Accordingly, this document is a "Getting started with *Extension of Knowledge Discovery Metamodel*" support to concisely explain how the D3.2 metamodels are used within the REMICS technology in general and BLU AGE® in particular.

The metamodels here evoked are the *KDM* and *ASTM* standards along with an innovation: the EKDM that has been built up for REMICS and is described in the D3.1 deliverable.

# Table of contents

# 1 Introduction

The ADM task force of the OMG has released several standards to carry out MDD reverse engineering activities and to support, more or less automatically, these activities in CASE tools.

In REMICS DoW, the strategy is to seek a maximum of compliance to OMG standards and, if possible, to influence these existing standards through innovation and progresses made in REMICS/WP3 in particular. For that, REMICS/WP3 reuses the KDM standard whose main and recognized implementation is that from *kdmanalytics.com*.

More recently, the ADM task force has released ASTM in January 2011 to fill a gap in the daily use of KDM for legacy code manipulation especially. As shown later in this document, ASTM complements KDM regarding code issues. This document emphasizes code manipulation through KDM, ASTM and **the concomitant and coherent use of both through a REMICS homemade metamodel named EKDM (Extension of KDM)**. These results from the absence of a clear and formal interrelation specification between KDM and ASTM, **a lack of experience/lessons learned in the daily use of these cutting-edge technologies as well**.

As primitive material, KDM, ASTM and EKDM cannot be used in a friendly way outside the scope of a tool. In REMICS, BLU AGE® from Netfective, a proprietary tool, is based on this metamodel suite. D3.1 document provides some information on KDM and ASTM along with the introduction and specification of EKDM. In D3.2 (this document), a focus is on the use of EKDM in the context of the COBOL programming language.

## 1.1 Terminology and Abbreviations

| | |
|---|---|
| ADM | Architecture-Driven Modernization |
| ASTM | Abstract Syntax Tree Metamodel |
| CASE | Computer-Aided Software Engineering |
| CRUD | Create, Read, Update and Delete |
| EKDM | Extension of KDM |
| EMF | Eclipse Modeling Framework |
| KDM | Knowledge Discovery Metamodel |
| MDD | Model Driven Development |
| OMG | Object Management Group |

## 1.2 Requirements traceability

D3.2 is part of WP3 whose original objectives are:

a) to define an integrated method for knowledge discovery to extract business value information from legacy including business models, components, implementation details and test specifications;

b) to specify the KDM extension to support the method;

c) to develop tools that supports the REMICS Recover process.

D3.2 fulfils the requirements as follows:

| | |
|---|---|
| a) | See D3.1. |
| b) | D3.1 gives an illustration of the utilization of EKDM. |
| c) | See D3.1. |

## 1.3 BLU AGE® REVERSE

BLU AGE® provides an open and extensible approach to achieve extraction, discovery and regeneration, from multiple types of legacy systems. BLU AGE® uses a model-based approach and a metamodel-driven methodology:

- Match different requirements systems modernization, data integration, etc.
- Use models operations and facilities: transformations, weavings, extractions, etc.
- Support methodology for defining extensions of the core metamodel and plug-ins to enable manipulating models, business rules, components/services, etc.

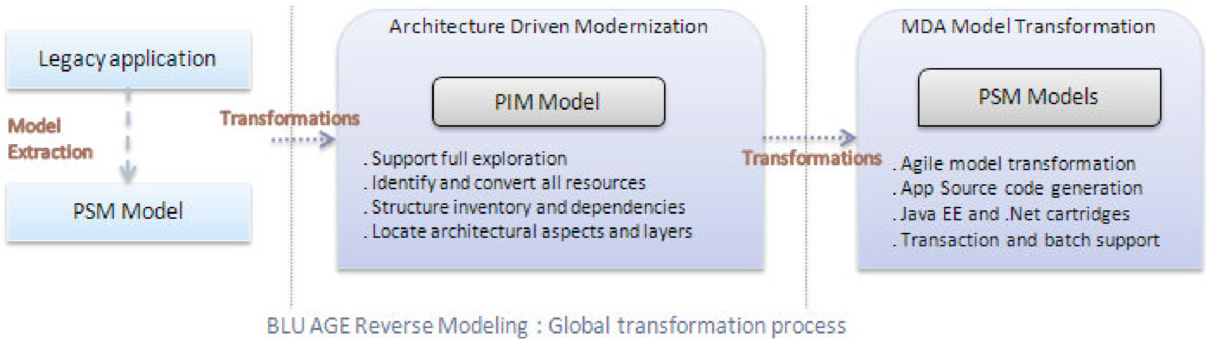
Legacy application
Model Extraction
PSM Model
Transformations
Architecture Driven Modernization
PIM Model
. Support full exploration
. Identify and convert all resources
. Structure inventory and dependencies
. Locate architectural aspects and layers
Transformations
MDA Model Transformation
PSM Models
. Agile model transformation
. App Source code generation
. Java EE and .Net cartridges
. Transaction and batch support
BLU AGE Reverse Modeling : Global transformation process

**Figure 1 – BLU AGE® REVERSE process**

Some trial versions and toy examples are downloadable from www.bluage.com/index.php?cID=blu_age_reverse_modeling_us.

# 2 Using EKDM with the BLU AGE® REVERSE engine

This section intends to show a simplified usage of EKDM based on a legacy code piece in Appendix I. This section is divided into three main parts: artifacts compilation, transformation of a specific model into a generic model and production of an EKDM-compliant model.

## 2.1 Artifact compilation

The BLU AGE® REVERSE process is composed of several well-codified stages. The first one amounts to extracting/collecting any information contained in the different artifacts (code, configuration files…) of the legacy application/information system of interest. Extraction is similar to a compiler at work. However, here, compiler outputs are models rather than executable code. Here is the processing chain: legacy artifacts -> lexical analysis -> syntactical analysis -> semantic analysis -> specific model -> generic pivot model. For that, grammars formalizing different languages are required.

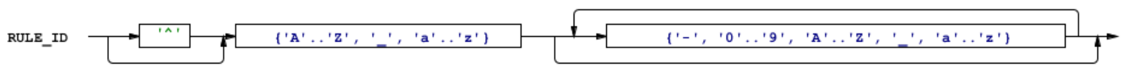
RULE_ID
'^'
{'A'..'Z', '_', 'a'..'z'}
{'-', '0'..'9', 'A'..'Z', '_', 'a'..'z'}

**Figure 2 – Example of terminal symbols**

Grammars are made up of terminal symbols (Figure 2), non-terminal symbols, axioms and production rules (Figure 3).

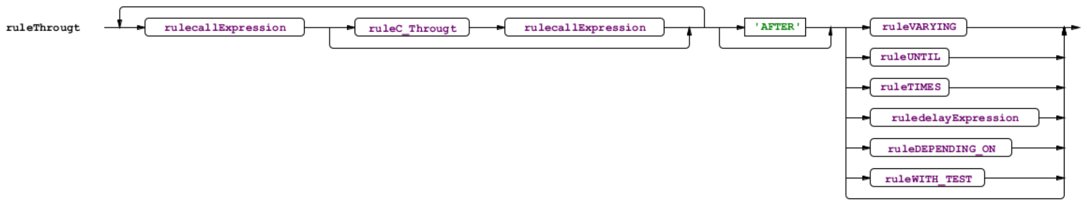

**Figure 3 – Example of the "Throught" (COBOL) production rule**

Specific models resulting from the initial compilation stages are Abstract Syntax Trees (ASTs). Any specific model is compatible with a homemade COBOL SASTM metamodel (see D3.1 about SASTM).

## 2.1.1 Specific model processing

### 2.1.1.1 Data

The COBOL program in Appendix 1 includes a "DATA DIVISION" section. This section itself contains "records" required in the program. Once the code below is parsed, one obtains the model in Figure 4.

```
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*
 01 COMPTEURS.
*
    05 COMPTEUR-A                     PIC 9 VALUE
             1.
```

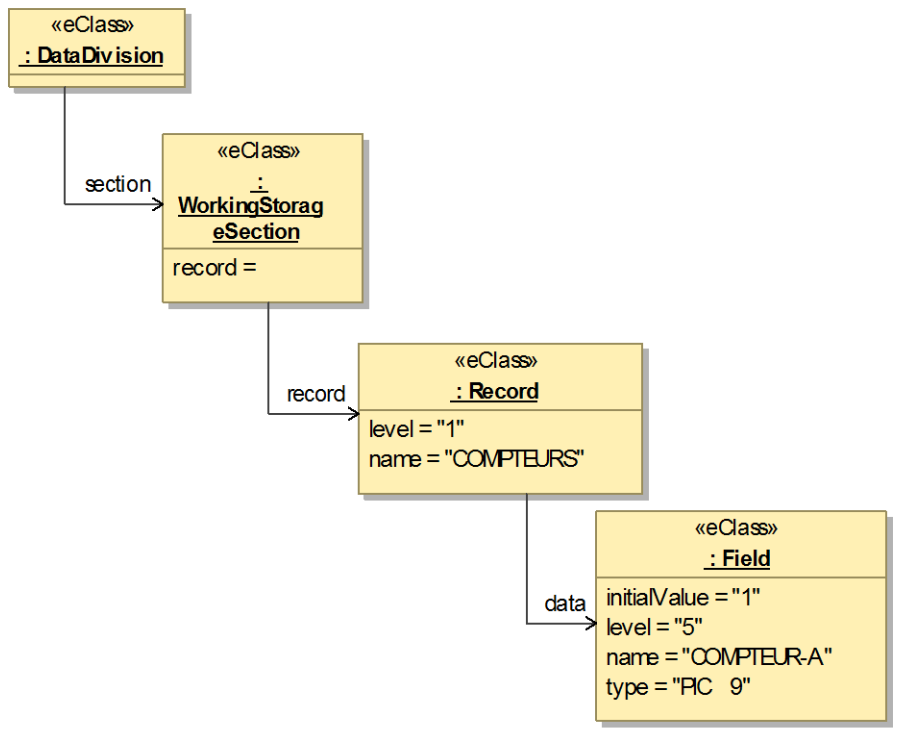

**Figure 4 – SASTM (partial) instance diagram of the "DATA DIVISION" section**

#### 2.1.1.2 UIs

In this document's example, UIs are also described in the "DATA DIVISION" section, more precisely, in a "SCREEN SECTION":

```
SCREEN SECTION
01 AFFICHAGE.
    05 CONSTANT.
      10 FILLER                         AT 01, 01   VALUE
                                          "Compteur:".
    05 SC-FIELD.
      10 SC-COMPTEUR-A                 AT 01, 11 PIC 9
           FROM COMPTEUR-A.
```

Once the code above is parsed, one obtains the model in Figure 5.

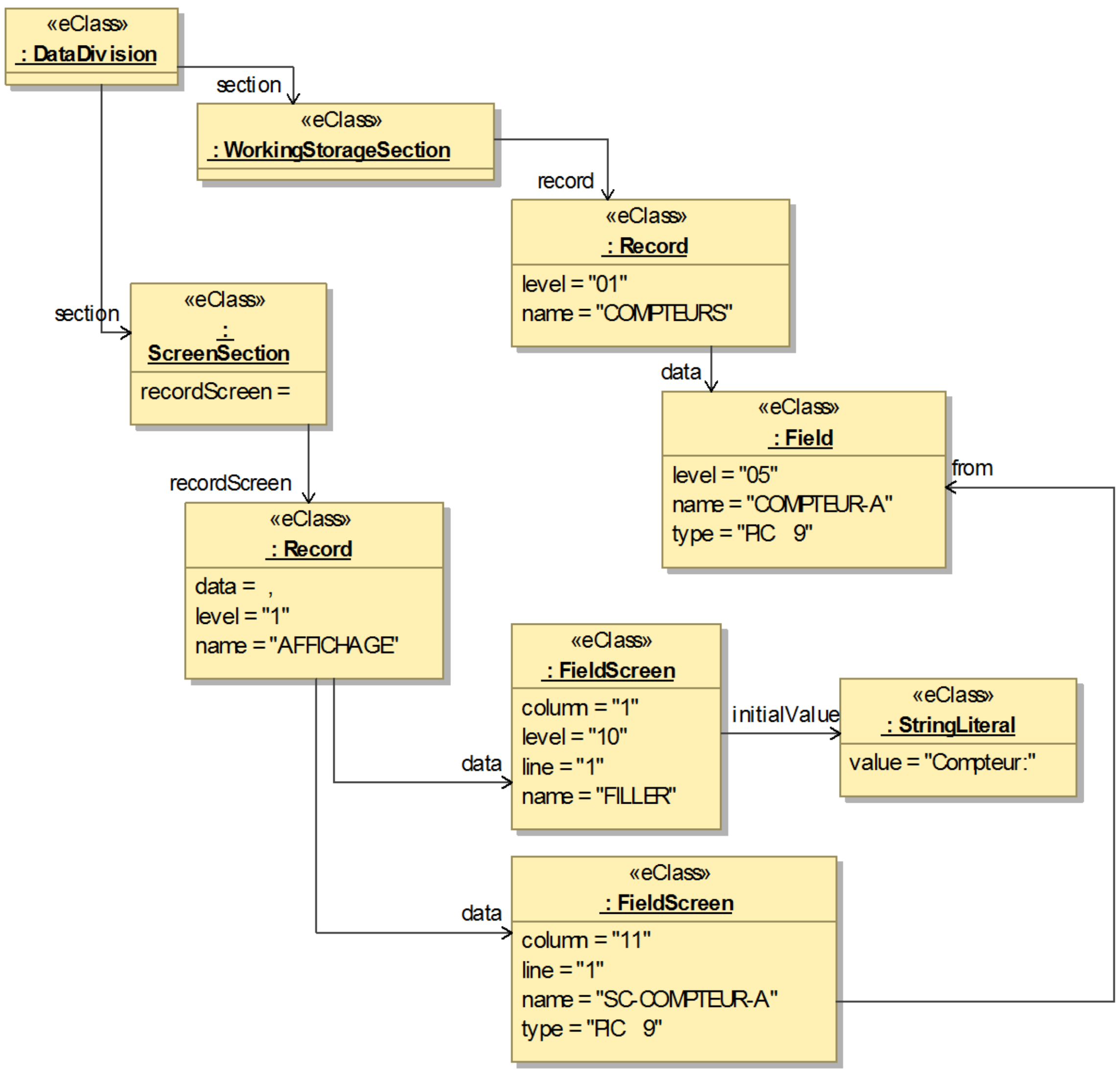

**Figure 5 – SASTM (partial) instance diagram of the "SCREEN SECTION"**

#### 2.1.1.3 Code

In COBOL, the "PROCEDURE DIVISION" contains procedural statements, for example:

```
A000-MAINLINE.
      PERFORM 1-INCREMENT THRU 1-INCREMENT-EXIT.
      PERFORM 1-DISPLAY THRU 1-DISPLAY-EXIT.
A000-EXIT.  EXIT PROGRAM.

 1-INCREMENT.
    ADD 1 COMPTEUR-A.
 1-INCREMENT-EXIT. EXIT.

 2-DISPLAY.

    DISPLAY BASE AFFICHAGE.
    DISPLAY COMPTEUR-A.
2-DISPLAY-EXIT. EXIT.
```

Once the code above is parsed, one obtains the model in Figure 6.

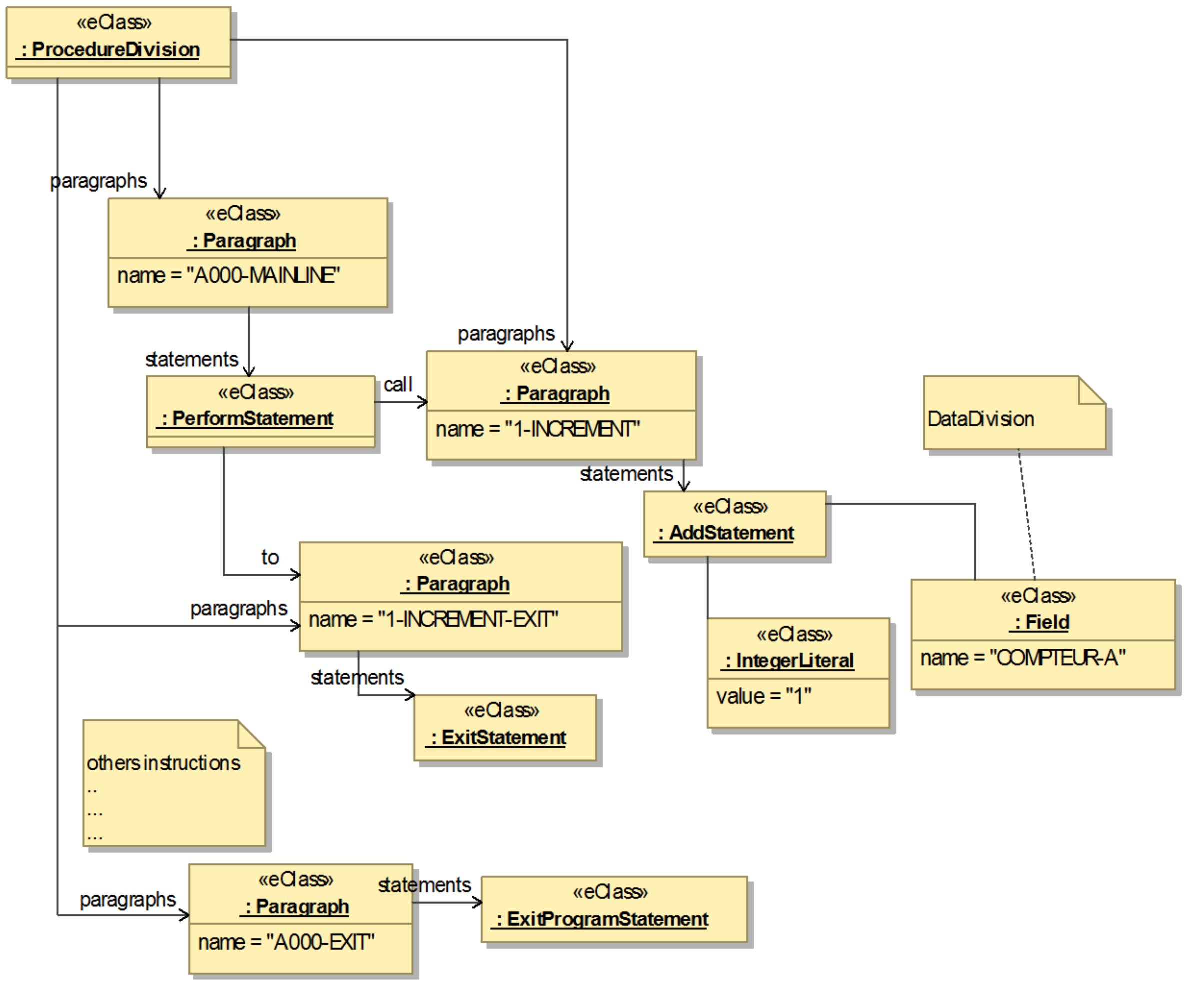

**Figure 6 – SASTM code instance diagram**

By definition, models computed in this first phase are legacy technology-specific, *i.e.*, they conform to a tailored SASTM metamodel. To move to legacy technology-independent models (GASTM), transformations are required. These generic models are called pivot models because they are **not** final models at the end of the processing chain. They aim at recomposing the overall coherence of the legacy stuff to create “semantic” models.

## 2.2 From SASTM to GASTM

To move to GASTM models, transformations are written with the help of a transformation language like ATL or a programming language like Java. These transformations are the core of the BLU AGE® engine. A key issue is the organization and the modularization of many simple transformations to avoid writing some of them legacy technology per legacy technology. We mean, some are fixed while some are dependent upon the deep nature of the given processed COBOL, say the dialect used in this document’s example.

### 2.2.1 Data generalization

Any “record” from the “DATA DIVISION” section is transformed in a “neutral” data structure with a type. Note that in COBOL, declarations and definitions are not distinguished. The result of such a transformation appears in Figure 7.

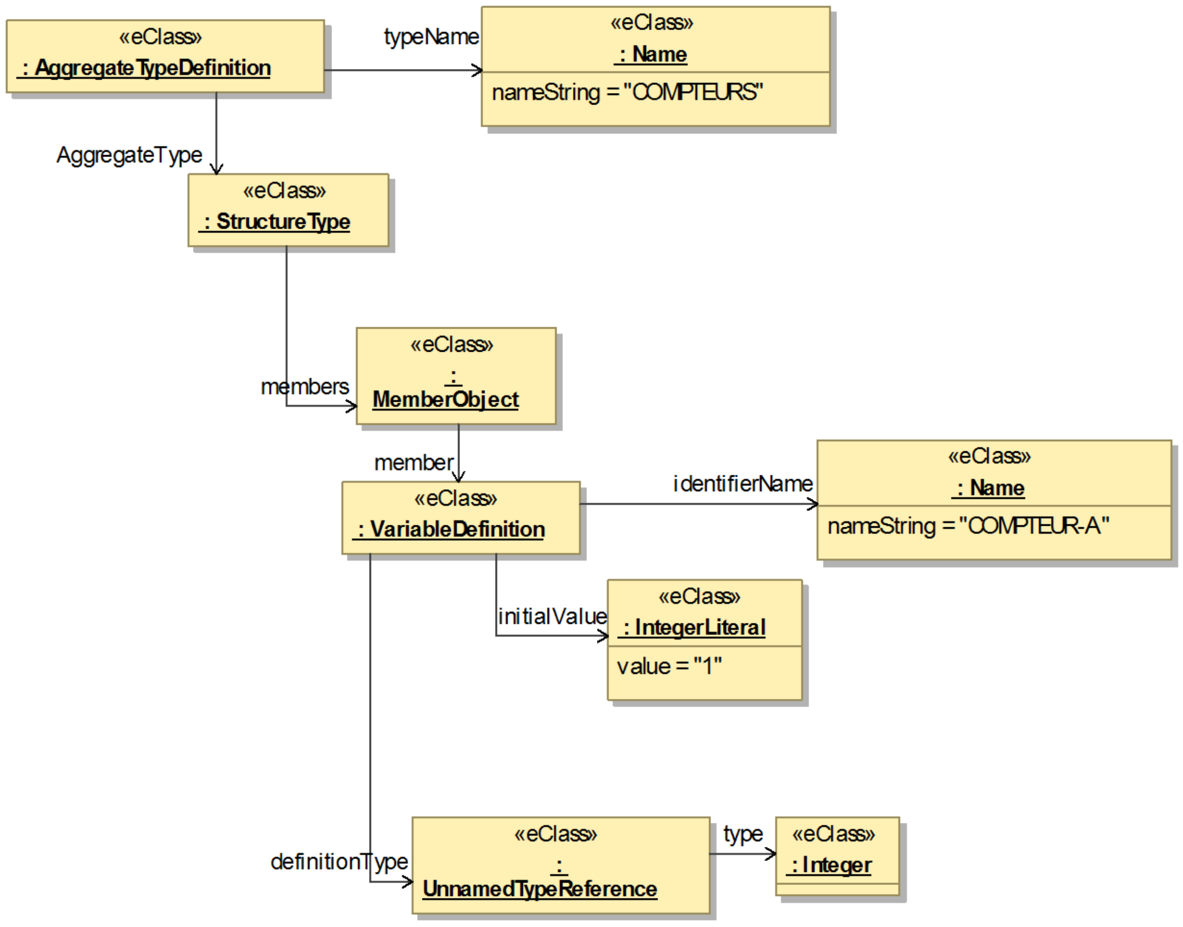

**Figure 7 – GASTM instance diagram resulting from the data transformation of the model in Figure 4**

### 2.2.2 UIs generalization

COBOL “displays” are subject to equivalent transformations.

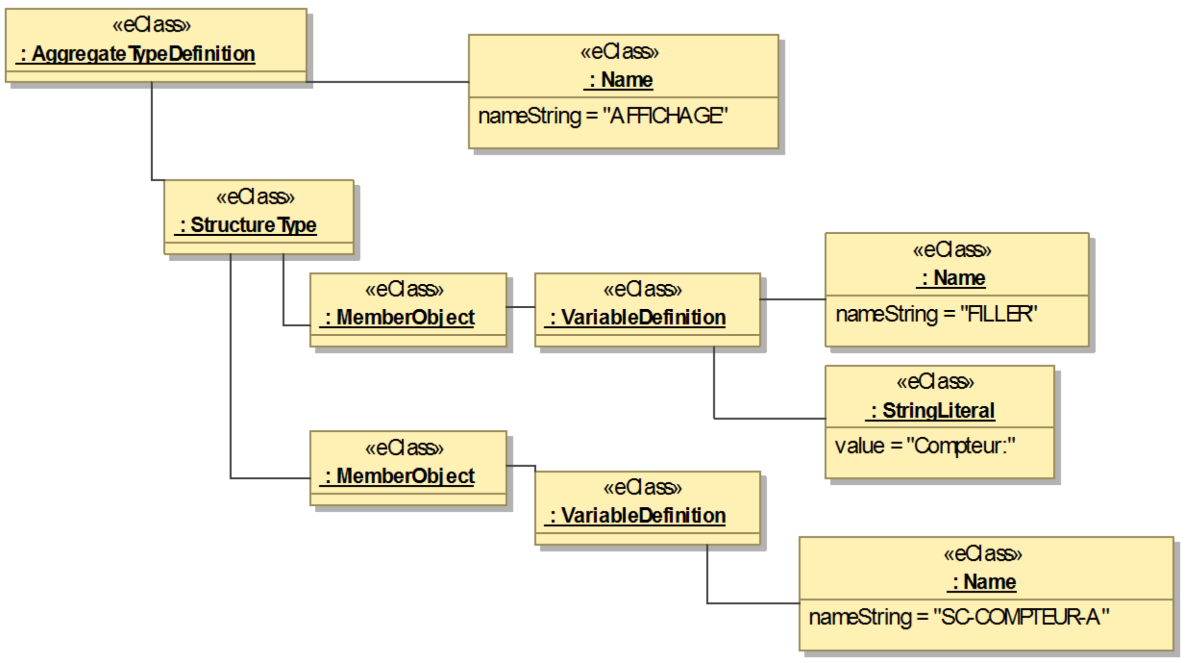

**Figure 8 – GASTM instance diagram resulting from the UIs transformation of the model in Figure 5**

There is a slight inconvenient (*i.e.*, a semantics loss) in the model appearing in Figure 8. Description information of screen properties and their mapping with data of the initial legacy code are not appearing in the model. More generally, this proves the lack of complementarity and articulation between ASTM on one side and KDM on another. At this stage, the legacy stuff is only represented by means of ASTM but persistent data and UIs typically aim at later being modeled as KDM first-class objects. We show in the next section and in the EKDM section of this document how to avoid semantics losses.

### 2.2.3 UIs extraction

A UI metamodel is immediately required to avoid information losses as discussed above. This metamodel is an extension of the *UI* KDM metamodel (see also D3.1 about KDM). This metamodel has been built up to have a generic description of textual UIs. Figure 9 shows how this metamodel allows the possibility to “keep” information about screen properties in relation with the data managed in UIs. For example, the “SC-COMPTEUR-A” variable is both viewed as a piece of a data structure (Figure 8) and a UI element (Figure 9).

How, apart from the two values (“SC-COMPTEUR-A”) of the *nameString* (Figure 8) attribute and *name* attribute (Figure 9), there is no explicit intelligible mapping between the two models. So, extending KDM is not only a matter of extraction but also of reconstruction of semantic dependencies between legacy artifacts. In this case, in GASTM, because of the generic nature of GASTM models, links between legacy-neutral data structure representations and legacy-neutral UI representations is required (see EKDM section below).

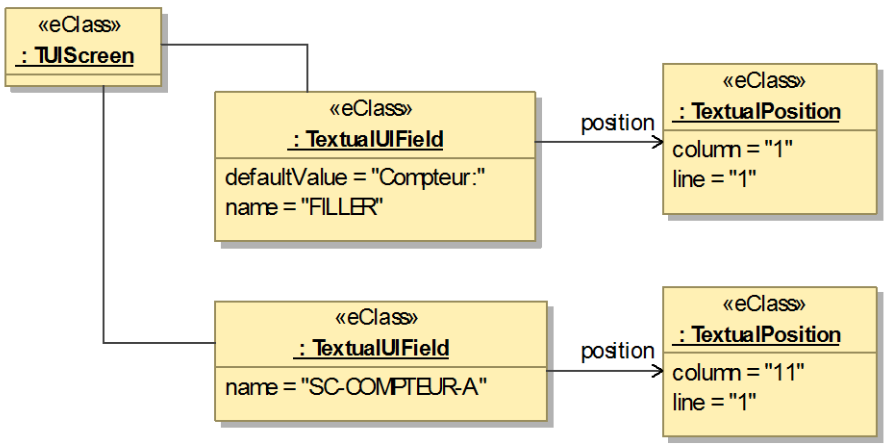

**Figure 9 – Instance diagram of a textual field model**

### 2.2.4 Code generalization, example of COBOL statements

COBOL statements are transformed into generic and easily interpretable representations (Figure 10). In Figure 10, there is an illustration with an “Add” COBOL action. A neutral form of the behavior of such a statement is stated.

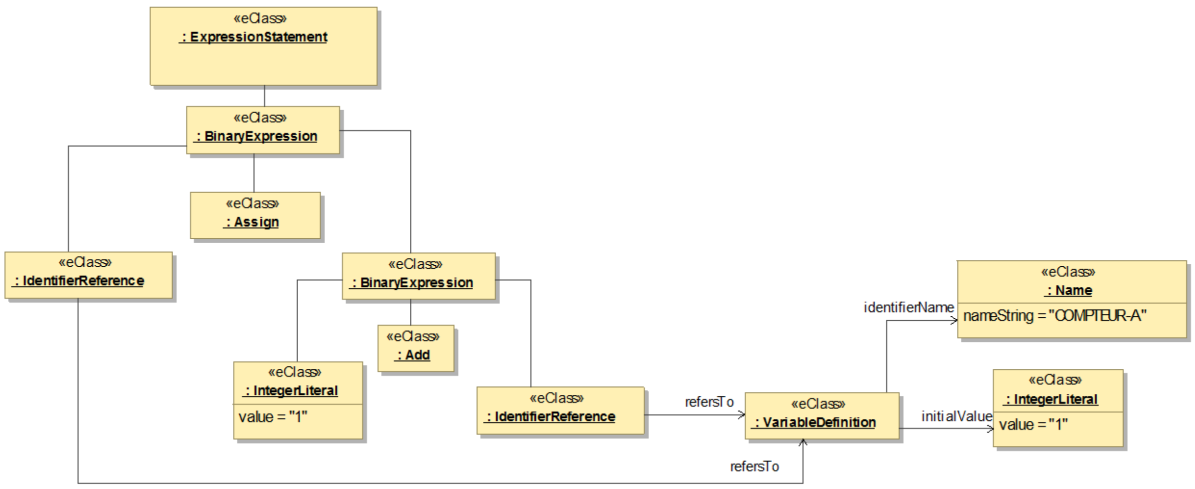

**Figure 10 – COBOL “Add” action from Figure 6, generalized**

At this stage, the availability of GASTM models creates a sound break with the legacy material. Even if most of this material is GASTM-compliant, a first extension of KDM named TUI (standing for Textual UI) is required to model UIs as layouts in old applications/information systems.

ASTM models in general provide a satisfactory overview that result from parsing all of the legacy artifacts. However, all concepts are from programming language notions like “expressions” and so on. This representation gives no value about data/control flows showing execution details. Beyond, more abstract views are required in KDM to augment the captured intelligence in the legacy application/information system: components/services, architecture and business rules. For that, EKDM aims at complementing the GASTM views by creating a gateway between the ASTM and KDM worlds.

## 2.3 EKDM

### 2.3.1 Separation of “macro” and “micro” code representations

While “micro” representations intend to give a rich set of details, “macro” representations emphasize key actions at the code level like CRUD actions for example and execution flows in general towards components/services and architecture patterns/forms.

### 2.3.2 GASTM encapsulation within KDM

**The followed strategy is the encapsulation of GASTM within KDM by means of EKDM.**

Using *MicroCodeModel* from EKDM that implements the *KDMModel* interface belonging to the *kdm* package of KDM is the major way by which encapsulation occurs. As an example, the GASTM model in Figure 9 is “surrounded” by a KDM model.

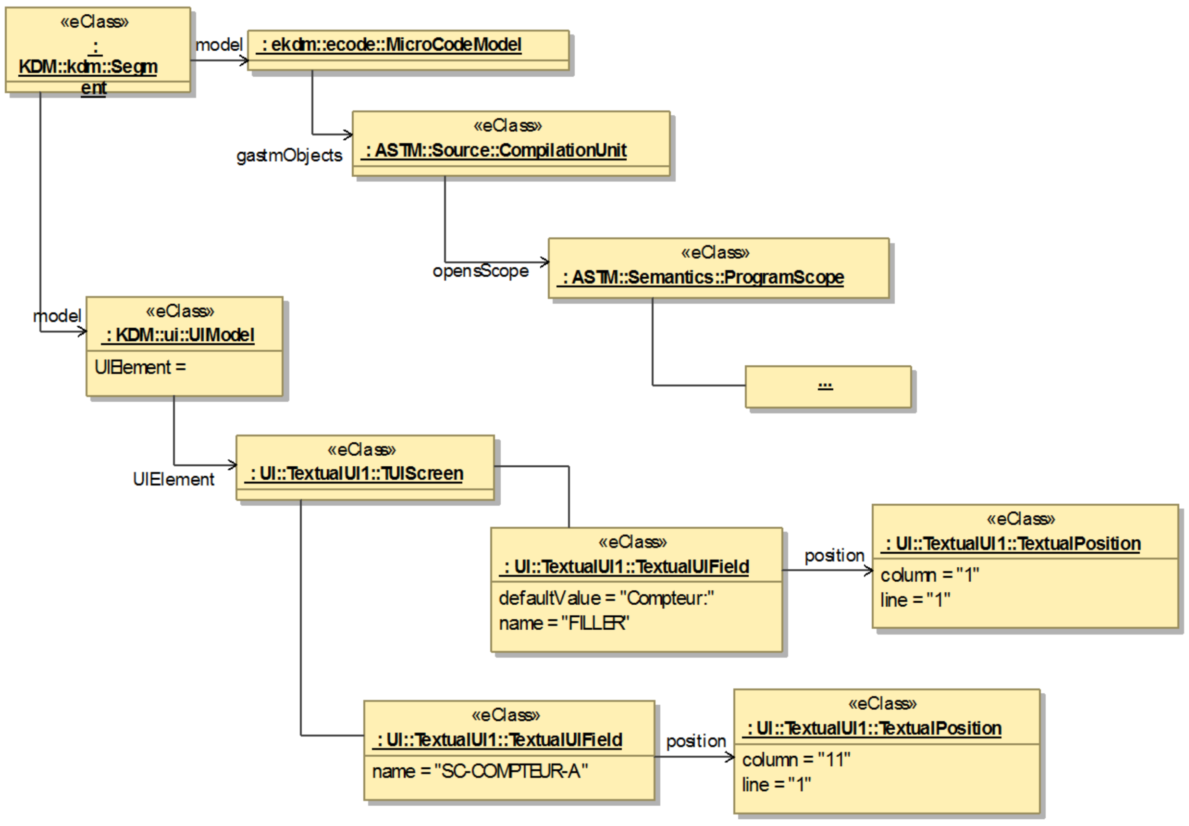
«eClass»
: KDM::kdm::Segment
model
: ekdm::ecode::MicroCodeModel
gastmObjects
«eClass»
: ASTM::Source::CompilationUnit
opensScope
«eClass»
: ASTM::Semantics::ProgramScope
...
model
«eClass»
: KDM::ui::UIModel
UIElement =
UIElement
«eClass»
: UI::TextualUI1::TUIScreen
«eClass»
: UI::TextualUI1::TextualUIField
defaultValue = "Compteur:"
name = "FILLER"
position
«eClass»
: UI::TextualUI1::TextualPosition
column = "1"
line = "1"
«eClass»
: UI::TextualUI1::TextualUIField
name = "SC-COMPTEUR-A"
position
«eClass»
: UI::TextualUI1::TextualPosition
column = "11"
line = "1"

**Figure 11 – ASTM model encapsulated within KDM**

The “micro” representation of the code is thus realized through ASTM with a ported access to KDM.

#### 2.3.2.1 “Macro” actions’ production

The goal of this production (transformation suite) is to have a more abstract view. Namely, EKDM *MacroActionElement* objects are created for each *Statement* object in ASTM. This allows the definition of semantic relationships between *MacroActionElement* objects and KDM objects. Figure 12 is an illustration.

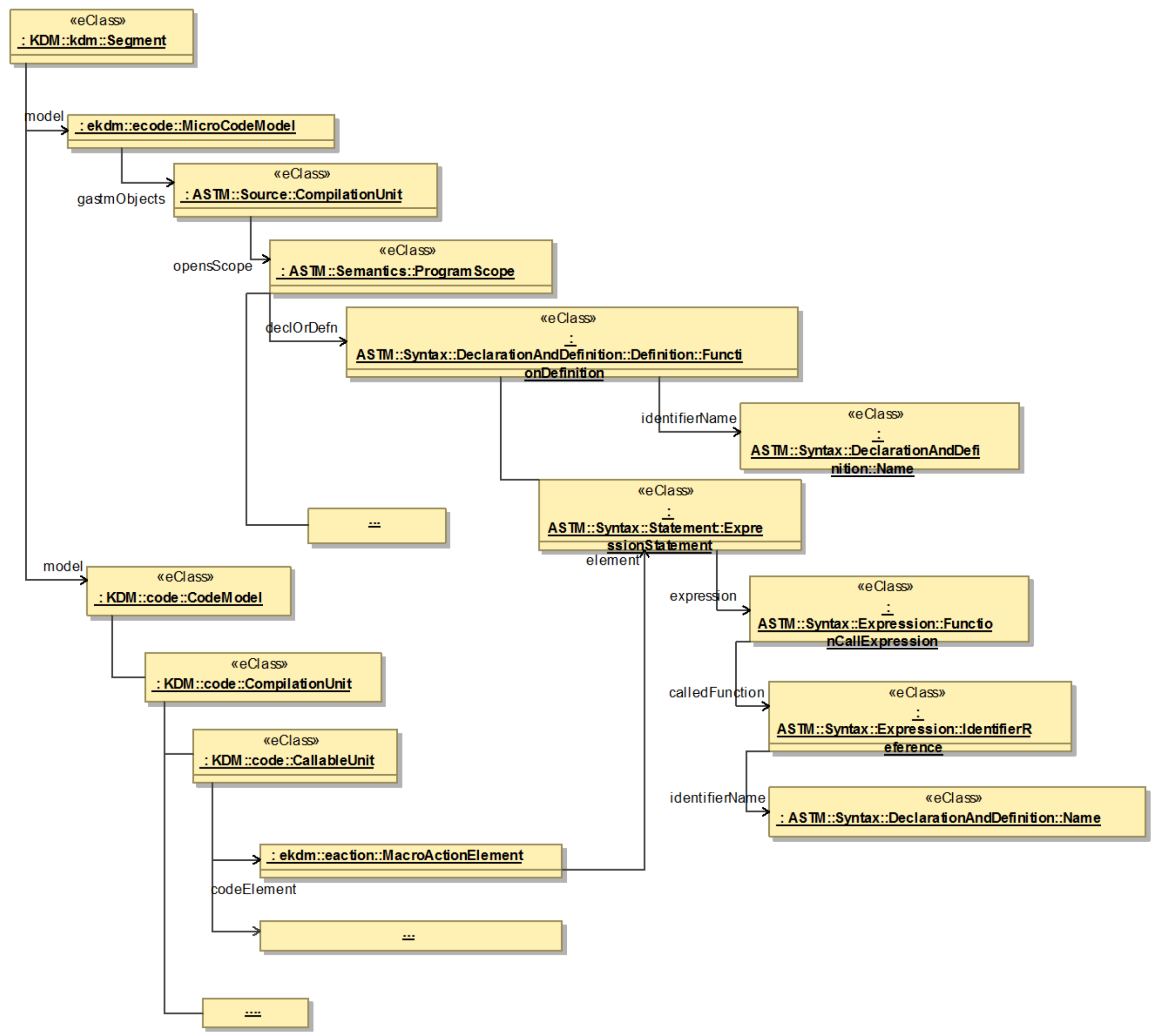
«eClass»
: KDM::kdm::Segment
model
: ekdm::ecode::MicroCodeModel
gastmObjects
«eClass»
: ASTM::Source::CompilationUnit
opensScope
«eClass»
: ASTM::Semantics::ProgramScope
declOrDefn
«eClass»
: ASTM::Syntax::DeclarationAndDefinition::Definition::FunctionDefinition
identifierName
«eClass»
: ASTM::Syntax::DeclarationAndDefinition::Name
...
«eClass»
: ASTM::Syntax::Statement:ExpressionStatement
element
expression
«eClass»
: ASTM::Syntax::Expression::FunctionCallExpression
calledFunction
«eClass»
: ASTM::Syntax::Expression::IdentifierReference
identifierName
«eClass»
: ASTM::Syntax::DeclarationAndDefinition::Name
model
«eClass»
: KDM::code::CodeModel
«eClass»
: KDM::code::CompilationUnit
«eClass»
: KDM::code::CallableUnit
: ekdm::eaction::MacroActionElement
codeElement
...
....

**Figure 12 – "macro" and "micro" action representation**

Mapping information is not treated here but in the next section.

### 2.3.3 Construction of mappings between code, UIs and data

Mappings are introduced from captures of information in the legacy application/information system of interest. Mappings are instances of the *UIMapping* and *DataMapping* from EKDM (see also D3.1). Mappings create the very last glue between several legacy elements partially and previously represented under several angles and metamodel perspectives.

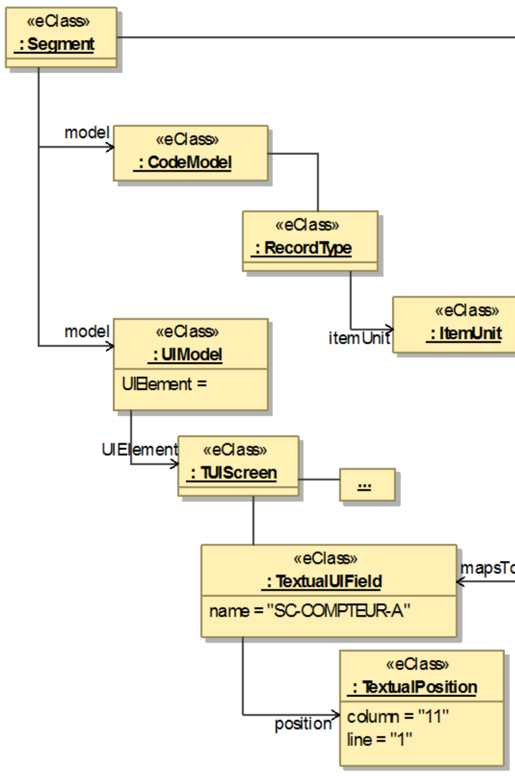

**Figure 13 – code/UIs mapping example**

### 2.3.4 Generation of relationships

From an analysis of the representation of “micro” actions (with possible manual intervention), a generation of “semantic” relationships between the different KDM objects occurs. The same occurs for “macro” actions and mappings afterwards.

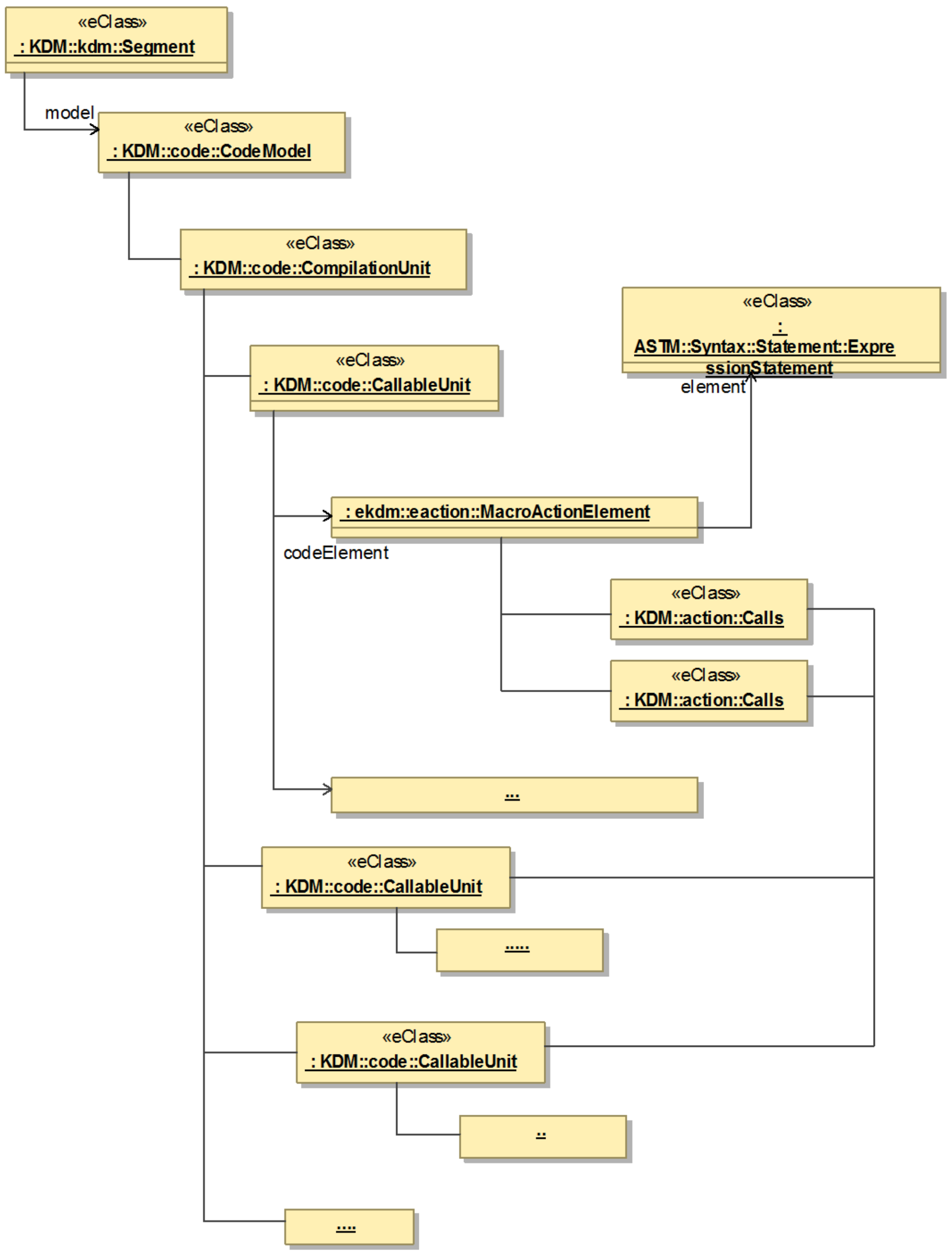

**Figure 14 – code "macro" representation with semantic relationships**

## Appendix I- Legacy code

```
 IDENTIFICATION DIVISION.
*
 PROGRAM-ID.    COMPTEUR.
*
 AUTHOR.        XXX.
*
 DATE-WRITTEN.  0001-01-01.
 ENVIRONMENT DIVISION.
*
 CONFIGURATION SECTION.
*
 SOURCE-COMPUTER.
*
     XXX.
 OBJECT-COMPUTER.
*
     XXXX.
 SPECIAL-NAMES.
*
     DECIMAL-POINT IS COMMA.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*
 01 COMPTEURS.
*
    05 COMPTEUR-A                     PIC 9 VALUE
             1.
SCREEN SECTION
01 AFFICHAGE.
    05 CONSTANT.
      10 FILLER                         AT 01, 01   VALUE
                                         "Compteur:".
    05 SC-FIELD.
      10 SC-COMPTEUR-A                 AT 01, 11 PIC 9
         FROM COMPTEUR-A.

PROCEDURE DIVISION.
A000-MAINLINE.
      PERFORM 1-INCREMENT THRU 1-INCREMENT-EXIT.
      PERFORM 1-DISPLAY THRU 1-DISPLAY-EXIT.
A000-EXIT.  EXIT PROGRAM.

 1-INCREMENT.
    ADD 1 COMPTEUR-A.
 1-INCREMENT-EXIT. EXIT.

 2-DISPLAY.

    DISPLAY BASE AFFICHAGE.
```