MODE-FE: A GRM/GDMO Parser and its API : Release 1.0 Reference Manual

Olivier Festor, Emmanuel Nataf, Laurent Andrey

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

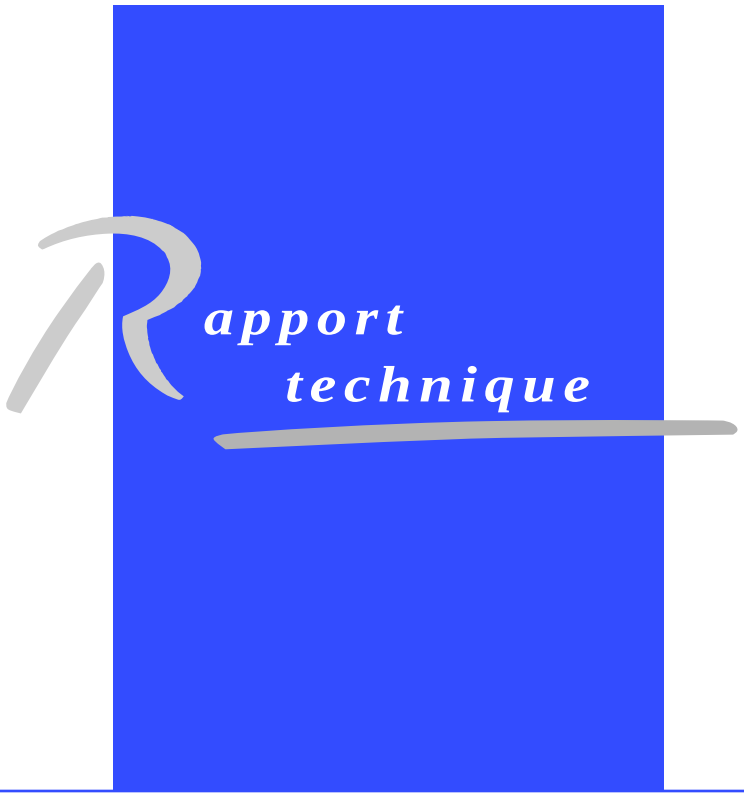
MODE-FE: A GRM/GDMO Parser and its API -Release 1.0- Reference Manual

Olivier Festor, Emmanuel Nataf, Laurent Andrey

N° 0190

Avril 1996

PROGRAMME 1



ISSN 0249-6399

MODE-FE: A GRM/GDMO Parser and its API
-Release 1.0-
Reference Manual

Olivier Festor, Emmanuel Nataf, Laurent Andrey

Programme 1 — Architectures parallèles, bases de données, réseaux et systèmes distribués
Projet RESEDAS

Rapport technique n°0190 — Avril 1996 — 115 pages

Abstract: MODE-FE (MODE Front-End) is part of the MODE (Managed Object Development Environment) prototype which is developed for integrating the use of Formal Description Techniques in an environment for building Management applications. MODE-FE provides a GRM (General Relationship Model) and GDMO (Guidelines for the Definition of Managed Objects) parser. The package offers specification checking and an Application Programming interface for manipulating loaded GDMO and GRM specifications. This document is the reference manual for release 1.0 of the MODE-FE (MODE Front-End) tool and its API.

Key-words: GDMO, GRM, MODE, Reference Manual

(Résumé : tsvp)

Unité de recherche INRIA Lorraine
Technopôle de Nancy-Brabois, Campus scientifique,
615 rue de Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY (France)
Téléphone : (33) 83 59 30 30 – Télécopie : (33) 83 27 83 19
Antenne de Metz, technopôle de Metz 2000, 4 rue Marconi, 55070 METZ
Téléphone : (33) 87 20 35 00 – Télécopie : (33) 87 76 39 77

MODE-FE: Un analyseur GRM/GDMO et son son interface de programmation
-Version 1.0-
Manuel de référence

Résumé :

MODE-FE (MODE Front-End) est une brique de l'environnement MODE (Managed Object Developement Environment) développé pour intégrer l'utilisation des méthodes de description formelle dans le développement d'applications de gestion de réseaux. MODE-FE est un analyseur de spécifications GRM (General Relationship Model) et GDMO (Guidelines for the Definition of Managed Objects). L'outil fournit un analyseur de spécifications ainsi qu'une librairie de programmation pour le traitement de spécifications GDMO et GRM. Ce document forme le manuel de référence de la version 1.0 de l'outil MODE-FE et de son interface de programmation.

Mots-clé : GDMO, GRM, MODE, Manuel de Référence

Chapter 1

Introduction

This document forms the reference manual of the **MODE-FE** (release 1.0) part of the MODE environment. MODE stands for **Managed Object Development Environment** and is a research prototype under development within the RESEDAS research group of the INRIA Lorraine and CRIN/CNRS. The MODE environment is designed for supporting various developments based on latest OSI standards for Management Information Modelling combined with Formal Description Techniques.

The MODE-FE (MODE Front-End) is the first released part of the MODE environment. It provides parsing tools and an application programming interface for both GRM (General Relationship Model) and GDMO (Guidelines for the Definition of Managed Objects) specifications.

MODE-FE provides:

- a Front-End for GRM specifications based on [CCITT.X.725 95],
- a Front-End for GDMO specifications based on [ISO-10165.4 92],
- an internal representation of all parsed GRM and GDMO templates,
- an Application Programming Interface allowing the query and manipulation of all parts of a loaded specification,

Mode-FE differs from available information modelling tools through:

- the General Relationship Model support,
- Management Platform independence through its API,
- both standard GDMO and extended GDMO support.

Mode-FE is distributed **FREELY** under the **COPYRIGHT** conditions as described in the Copyright & Contact chapter of this document and in the `COPYRIGHT` file provided with the package. Mode-FE is not a closed environment, nor it is a commercial one. Even if stable and complete, it aims at allowing people to experiment new features in Management Information Modelling.

Within this document, the architecture of the MODE-FE library is presented, its API detailed and examples of its use given.

The remainder of this document is organised as follow. Chapter 2 provides an overview of the tool. Chapter 3 states the compliance of the parsed notation to the standards and details the semantics checked by the tool. Chapter 4 contains a detailed description of the MODE-FE package and its use. The document is completed with a conclusion in chapter 6 and a presentation of planned extensions (chapter 7. In an appendix, the reader will find a detailed description of the Application Programming Interface provided within the MODE-FE as well as an example of its use.

Chapter 2

MODE-FE overview

As already stated in the introduction, MODE-FE is part of the MODE environment and is responsible for the parsing, storage and access interface to both GRM and GDMO specifications. According to this, MODE-FE is built on a layered architecture as depicted in figure 2.1.

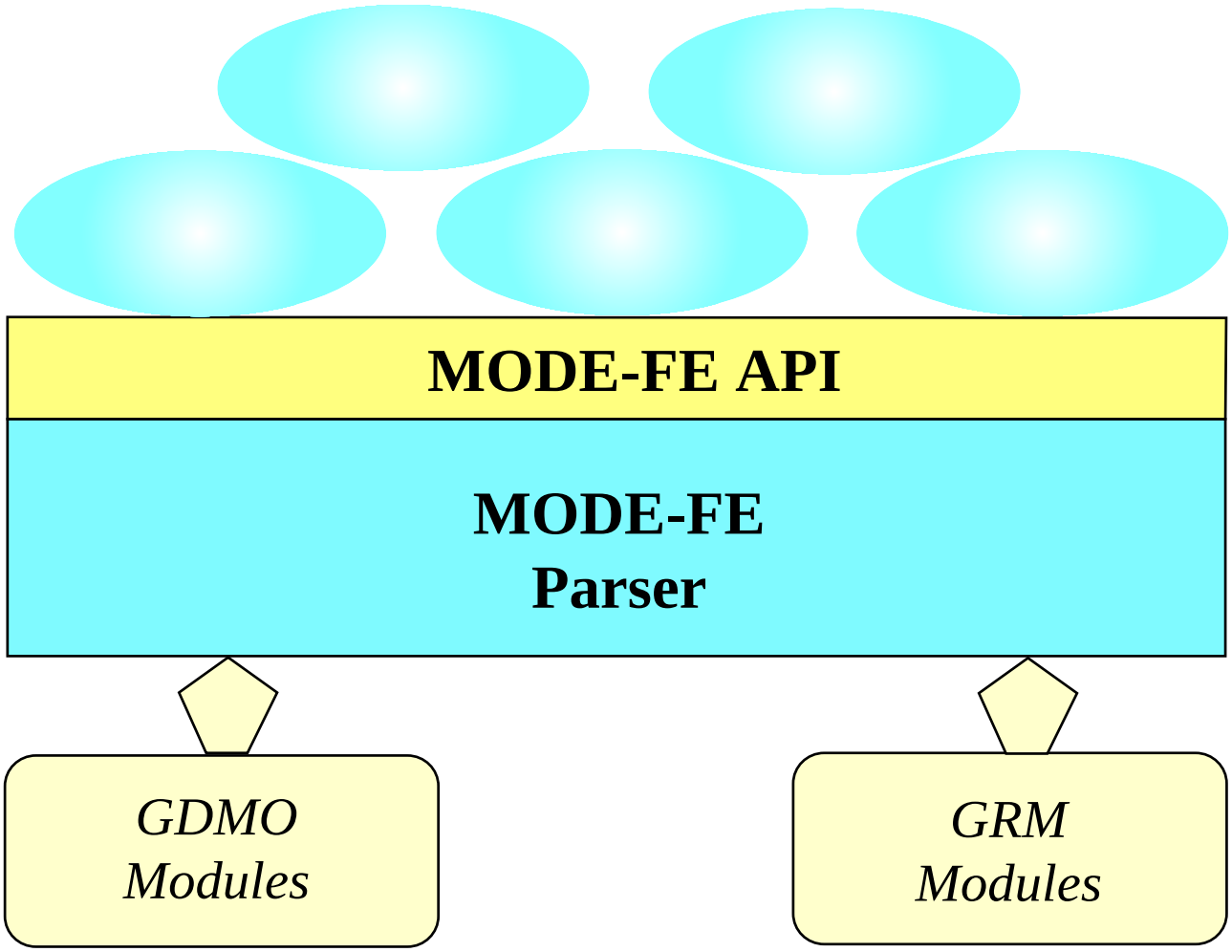


Figure 2.1: The MODE-FE layers

MODE-FE contains two layers. The lower layer allows to load GRM and GDMO specifications through a parsing module. Modules are loaded one by one and the contained specifications stored within a repository accessible via the MODE-FE API. The API makes the link to the upper layers of the environment where one can build any desired application module, e.g. a specification editor, a code generator, ... This version of the

package provides the library and its API and one small application module which performs pretty-printing for both GRM and GDMO specifications.

The whole MODE-FE library and applications have been developed in standard C++ according to the ELLEMTEL programming rules [Nyquist 92]. It does not use any proprietary lib, is portable over multiple platforms and compilers and support is provided by the authors.

Chapter 3

Standard Compliance

As the toolkit is used within our group to link standard Management Information Models with Formal Methods, we use the behaviour description part of the templates to put the formally described behaviour parts without affecting the standard notations.

This does not affect the compliance to the standard notations.

However, we have slightly extended the GDMO and GRM notations with three minor features. These features are:

- the ability to associate a module identifier to a set of GRM/GDMO specifications, i.e. a file,
- a behaviour clause associated with managed objects classes allowing the association of behaviour templates at the Managed Object level,
- a class keyword in the operation mapping part onto a `CREATE` operation of GRM specifications allowing the differentiation of a class identifier from the associated parameters.

These differences are described below.

3.1 Module identifiers

Each specification file may be associated with a module identifier. This module identifier is used at the semantics checking level to identify the origin of a label or a definition. Usually the module identifier is the identifier associated with a standard e.g. *"Recommendation X.721:1992:"* which identifies labels defined in the X.721 recommendation.

If required, the module identifier must be specified at the beginning of a file according to the following syntax:

MODULE " *<identifier>*";

<list of gdmo and grm specifications>

<end of file>

The specification of the module identifier is optional in the MODE_FE framework.

Thus in the X.721 recommendation, the associated GDMO file should start with the following text:

MODULE *"Recommendation X.721:1992:"*";

...

3.2 GDMO extensions

We have added the support for behaviour specification at the managed object class specification level. There, the specifier can in addition to the standard define a behaviour clause like in many other templates of the GDMO and GRM framework.

Thus the new syntax associated to a GDMO managed object class template is as described below.

```
<class-label>  MANAGED OBJECT CLASS
[  DERIVED FROM <class-label> [ ,<class-label>]* ;
]
[  CHARACTERIZED BY <package-label> [ ,<package-label>]* ;
]
[  CONDITIONAL PACKAGES <package-label> [ ,<package-label>]*
                                    PRESENT IF  condition
                     [ ,<package-label> [ ,<package-label>]*
                                    PRESENT IF  condition]* ;
]
% added to the standard GDMO notation
[  BEHAVIOUR <behaviour-label>;
]
 REGISTERED AS <object-identifier> ;
```

This does not affect standard GDMO support within the parser.

3.3 GRM extensions

We have added one keyword to the relationship mapping template. There we have extended the system operation part of the operation mapping which concerns the CREATE operation with the CLASS keyword as illustrated below:

```
% old syntax
<system-management-operation> ->
...
| CREATE [<class-label>] [<parameter-label>]*
...

%new syntax
<system-management-operation> ->
...
| CREATE [ CLASS <class-label>] [<parameter-label>]*
...
```

The introduction of this keyword allows the Front-End to distinguish a class label from a parameter label at the syntax parsing level of a specification and is thus not reported at the semantics check which acts now in a non tricky way to treat this part of a specification.

3.4 Known limits

The MODE-FE parser parses all GRM and GDMO files and provides a semantics checker a a separate library. It is not yet linked to an ASN.1 compiler and thus is not able to perform all possible semantics checks.

MODE-P does not make any tests on ASN.1 types. Since several ASN.1 parsers are already provided, the authors do not want to build yet another ASN.1 parser. However, since ASN.1 type identification is required for several semantical checks within GDMO and GRM specifications, a link to an existing ASN.1 compiler is planned.

As we do not have the latest addendum to the standards related to the GDMO and GRM notations, some small changes to the parser may be required. These are easily feasible and will be made very soon.

Chapter 4

Package Description and installation

4.1 Contents

The MODE-FE environment is provided with:

- the ModeFE pre-compiled library,
- all header files,
- the TeX Pretty-Printing Library,
- a sample application that parses GRM and GDMO specifications and builds the corresponding TeX file that can be automatically included in a LaTeX document.

The toolkit is provided as a tar zipped file (ModeFE_XXX_YYY.tar.gz where XXX is the identifier of the operating system on which the library is compiled and YYY the name and the version of the compiler).

unfolding the package through the

```
<prompt>gunzip ModeFE_XXX_YYY.tar.gz
<prompt>tar -xvf ModeFE_XXX_YYY.tar
```

which create the following directory architecture and extract following files:

The `COPYRIGHT` and `COPYRIGHT.FRENCH` files contain the copyright information for the package. The `README.LATTEST` file contains latest information over the package such as version and a short description of the package. The `CHANGES` file provides information on evolutions from one version to another.

The `Doc` directory provides all reference manuals required for the package. In the first release of the package, only this report is available.

In the `src` directory, you will find a `Main.cc` which is a sample main file for parsing a module and building a formatted output in TeX of the content of the specification.

All executable programs are generated in the `bin` directory.

The `mak` directory contains two makefiles, one for GNU-make and one for the standard make command.

A precompiled library is provided for each operating system in the corresponding libXXX directory.

4.2 Compiler Requirements

The MODE-FE tool was originally built on a SunOS 4.1.3_U1 operating system using the GNU G++ compiler, the Flex++ lexical analyser generator and Bison++ grammar analyser generator.

The versions of these tools are:

- g++ version 2.6.3
- flex++ version 2.3
- bison++ version 2.2

```
MODE/
        COPYRIGHT
        COPYRIGHT.FRENCH
        README.LATTEST
        CHANGES
        /Doc/
                ModeFE_lib.ps
        /src/
                Main.cc
        /lib/
                /SunOS4.1.3_U1/
                        libModeFE.a
                        libModePP.a
                /HPUX/
                        libModeFE.a
                        libModePP.a
                /Solaris/
                        libModeFE.a
                        libModePP.a
                /IRIX/
                        libModeFE.a
                        libModePP.a
        /bin/
        /mak/
                Makefile
```

In its first version, the ModeFE library is provided as a pre-compiled library and thus, availability on your platform of flex++ and bison++ is not required.

We are porting the tool on several environments and several compilers. The library is currently available on following platforms and compiled with the following compilers:

Operating System	**Compiler**
SunOS Release 4.1.3_U1	GNU g++ version 2.6.3 & 2.7.1
Solaris (SUN OS 5.5)	GNU g++ version 2.6.3
HPUX	GNU g++ version 2.7.0 & 2.7.1
SGI IRIX 5.3	GNU g++ version 2.7.0 & 2.7.1

The first package is provided with the SunOS 4.1.3_U1 library compiled with g++ 2.6.3, the HPUX library compiled with g++ version 2.7.0 and the SGI IRIX library compiled with g++ version 2.7.0.

Other libraries are provided freely on demand.

4.3 Compiling the example

The example provided with the library allows the parsing of a specification file and generates a TeX file which contains the formatted specification according to the OSI way of providing formatted specifications.

The invocation synopsis of the program is:

```
Main <grm-gdmo-file> -o <output-TeX-file>
```

The program must be compiled first. To this end go to the make directory of MODE and type **`make xxx`** where xxx corresponds to your system. Possible values for the system are:

- sun
- hp

- sgi
- solaris

The code of the sample program is given in appendix B of the report together with details on the use of the Mode-FE Library in applications.

Chapter 5

Copyright & Contact

5.1 Contact

The MODE toolkit is still under development. Several back-ends for MODE-FE are either in development phase or β-test and some improvements to access the repository will be available in a very short time.

We are maintaining the MODE-FE library. So feel free to send us comments, bug reports. We will do our best to provide a stable and usage friendly tool.

Please send:

- bug reports,
- comments,
- any suggestions,

to one of the authors. We will respond in the shortest delay.

Contact person
Dr. Olivier Festor
RESEDAS Research Group
INRIA Lorraine
Technopole de Nancy-Brabois
- Campus scientifique -
615, rue de Jardin Botanique - B.P. 101
54600 Villers Les Nancy Cedex
France
E-mail: festor@loria.fr
Tel: (33) 83.59.20.16
Fax: (33) 83.27.83.19
URL:http://www.loria.fr/ festor

A WWW page is maintained for MODE. The URL is:

http://www.loria.fr/exterieur/equipe/resedas/MODE.html

On this page you will find all new libraries developed for the the tool, new accessible applications, documentation, technical reports and white papers concerning planned extensions. You will also find many links to other network management pages from general information to companies which provide products for OSI and Internet Management.

5.2 Copyright

The ModeFE library is distributed freely under the conditions specified in the COPYRIGHT file provided with the library. The reader will find below the content of the Mode-FE copyright.

The Software ModeFE (c) INRIA 1995 in its release 1.0 of the 16/04/1996, hereafter referred to as "The SOFTWARE".

The SOFTWARE has been designed and produced by O. Festor and E. Nataf, the researchers of the RESEDAS project, a research project of the National Computer And Automatics Institute (INRIA), Domaine de Voluceau, Rocquencourt, 78153 Le Chesnay Cedex.

INRIA holds all the patent rights concerning the SOFTWARE. The SOFTWARE has been registered at the Agency for the Protection of Programmes (APP) under the number ???????.

Foreword

The SOFTWARE is currently being developed and INRIA wishes for it to be used by the scientific world so as to test, evaluate and continually update it.

To this end, INRIA has decided to distribute the prototype of the SOFTWARE by FTP, in an Object-code form.

a) Extend of the rights granted by INRIA to the user of the SOFTWARE:

INRIA freely grants the right to use, modify and integrate the SOFTWARE in another program.

b) Reproduction of the SOFWARE

Clauses 9 and 10 of the Berne agreement for the protection of literary and artistic works (Union of Berne) respectively specify in their paragraphs 2 and 3 authorising only the reproduction and quoting of works on the condition that:

- *" This reproduction does not adversely affect the normal exploitation of the work or cause any unjustified prejudice to the legitimate interests of the author",*
- *"That the quotations given by way of illustration and/or tuition conform to the proper uses and that it mentions the source and name of the author if this name features in the source",*
- Any use or reproduction of the software items and/or documents exclusively owned by INRIA and carried out to obtain profit or for commercial ends being subject to obtaining the prior express authorisation of INRIA.
- Any commercial use made without obtaining the prior express agreement of the INRIA would therefore constitute a fraudulent imitation.

c) Information feed-back

Any user of the SOFTWARE shall send his comments on the use of the SOFTWARE to the INRIA at (email : festor@loria.fr).

d) Guarantees:

Note that the SOFTWARE is a research product currently being developed.

INRIA disclaims any responsibility in any way in any instance of being obliged to put right any possible direct or indirect damage sustained by the user.

Chapter 6

Conclusion

In this document we have presented the MODE-FE tool and its API. MODE-FE is the first module of the MODE environment. This module provides features for parsing GRM and GDMO specifications and enables the development of application modules over its API.

In our research group several application modules are under development over the Mode-FE toolkit. Thus many new building blocks will be provided soon. Every person interested in the OSI Management Information Modelling field is encouraged to use the toolkit.

Chapter 7

Planned extensions

Several extensions are planned for the MODE-FE toolkit. Not all are definitively retained yet. Within these extensions we already identified:

- **Harmonised and safe API:** some of the access methods must be harmonised in order to facilitate the usage of the API and some minor bugs may have to be fixed.
- **More powerful Access methods** to the library are under construction.
- **Extended GRM support:** we have defined several extensions to the GRM notation to allow both more expressive power in GRM specification and a more automated support for GRM behaviour in Management applications. The extended GRM will be supported soon.
- **Management applications:** which exploit information provided by MODE-FE are currently under consideration (many applications), development (some applications) or test (a few applications) in our group.
- **Pretty printer:** this facility is already implemented into the MODE environment and allows to generate TeX files which are conform to the ITU style of Managed Object catalogs. This feature is provided as a separate library and is built on top of the MODE-FE library. This library will be extended very soon with many customisation facilities.
- **Formal Description Techniques:** Support of FDTs oriented modules are under development in the group.
- **Conformance requirements generator:** this facility allows the interactive generation of MOCS and GRM Conformance Statements proformas from the specification. We have already started working on this module and it should be available before spring 1996.
- **Code generator:** to allow the use of the front-end within several management platforms.
- **Specification comments treatment and "As is" formatted printing:** We are currently extending the parser in a way that the comments written in a row GRM or GDMO specification are kept in the parser and a pretty printer for raw specifications (with comments and inline specifications) is under development.
- ...

Bibliography

[CCITT.X.725 95] Comité Consultatif International Télégraphique et Téléphonique (CCITT), *"Information Technology - Open Systems Interconnection - Structure of Management Information - Part 7: General Relationship Model"*, Draft International Standard, CCITT.X.725, June 1995.

[ISO-10165.4 92] International Organization for Standardization (ISO), *"Structure of Management Information - Part 4: Guidelines for the Definition of Managed Objects"*, International Standard, ISO-10165.4, January 1992.

[Nyquist 92] E. Nyquist et Henricson M. *"Programming in C++: Rules and Recommendations"*. Ellemtel Telecommunication Systems Laboratories, Sweden, 1992.

Appendix A

The repository API

In this chapter, the API is presented in detail. The chapter is divided in three sections. The first one details all information classes which store information related to a GRM or a GDMO specification as well as all basic information classes. The second section presents the misc classes used by the parser together with additional functions provided in the library. The last section is concerned with the presentation of all list classes used in the repository.

A.1 Basic Information Classes

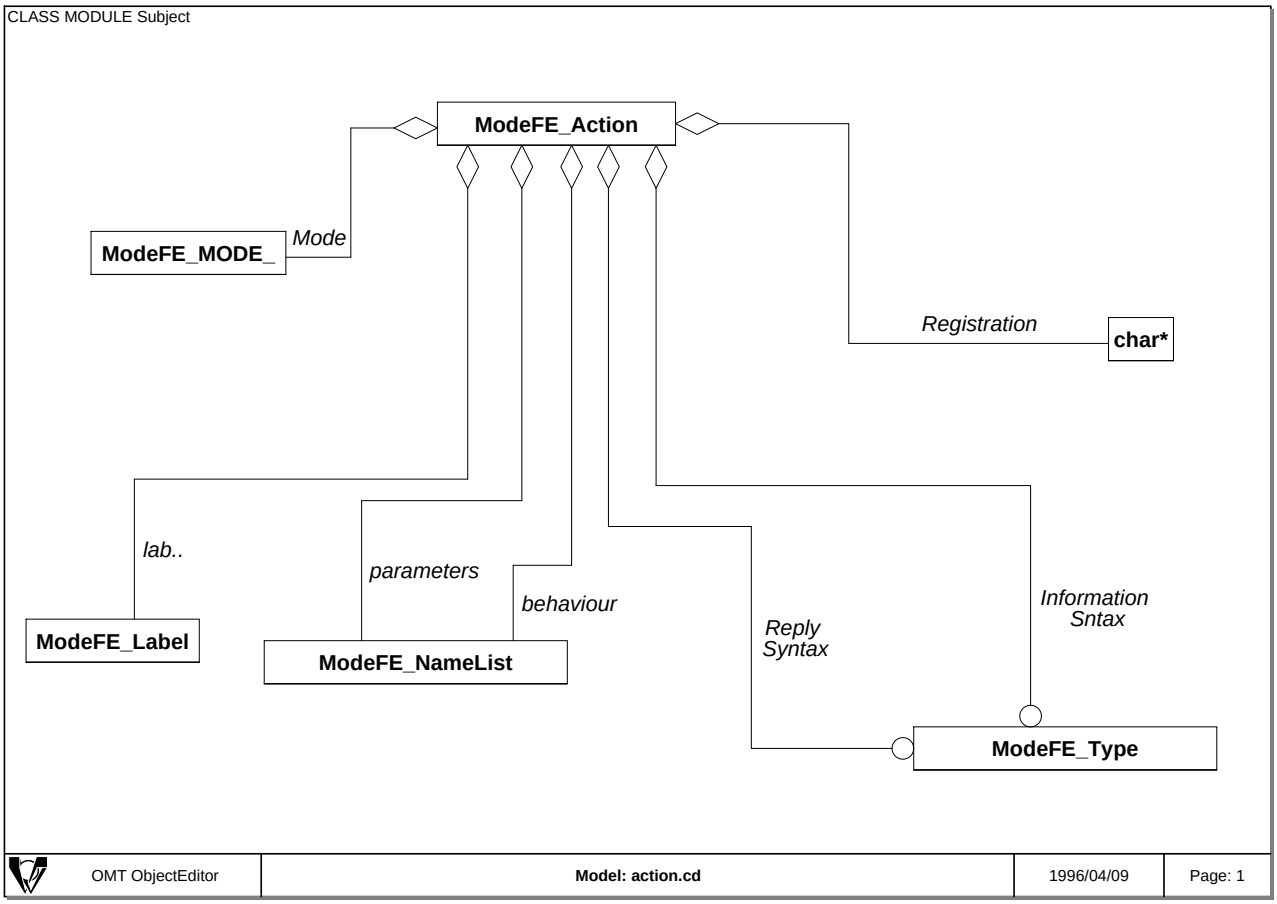
In this section, all information classes used within the MODE-FE library are presented. These C++ classes are presented in alphabetical order. For each class, its purpose is detailed, an OMT-like diagram of its architecture (its components and links to other classes) given and all public methods detailed.

A.1.1 The ModeFE_Action Class

Purpose

The ModeFE_Action Class stores information related to the specification of a GDMO Action type. It contains the action label, its registration identifier, the list of associated behaviour labels, the list of associated parameters, the information syntax if specified, the reply syntax if specified and the invocation mode (confirmed or not).

Architecture



Available methods

```
ModeFE_Action (ModeFE_Label* ,
               char * ,
               ModeFE_NameList *,
               ModeFE_MODE ,
               ModeFE_NameList *,
               ModeFE_Type ,
               ModeFE_Type );
```

Basic constructor for an action object. The parameters are:

1. the label associated with the action specification,
2. the registration identifier of the definition without the "{" and "}" brackets,
3. the list of associated behaviour labels,
4. the actions mode. Possible values are :
   - `CONFIRMED_`
   - `NOTCONFIRMED_`
5. the list of associated parameter labels,
6. the information syntax (`NULL` if none),
7. the reply syntax (`NULL` if none).

```
ModeFE_Action ();
```

Constructor for the ModeFE_Action class which creates an empty ModeFE_Action definition.

```
ModeFE_Action (const ModeFE_Action&);
```

Copy constructor of the ModeFE_Action class. Returns no element in the current version of the library.

```
~ModeFE_Action ();
```

Destructor of the ModeFE_Action class. Deletes all contained elements.

```
ModeFE_Label* GetLabel();
```

Returns a pointer to a copy of the action type label. If no label is associated, returns a pointer to an empty ModeFE_Label object.

```
int SetLabel(ModeFE_Label*);
```

Assigns a new value to the action type label. Returns `OK` if the operation was successful. Returns `ERROR` otherwise.

```
char* GetRegistration();
```

Returns a pointer to a copy of the action type registration identifier. The identifier is provided as a string and does not contain the "{" and "}" brackets.

```
void SetRegistration(char*);
```

Assigns a new registration identifier to the action specification. The registration identifier should not include the start and stop brackets ("{","}").

```
ModeFE_NameList* GetBehaviours();
```

Returns the labels of all behaviour definitions associated with the action type. If no behaviour is associated, then this methods returns an empty list.

```
int AddBehaviour(ModeFE_Label *);
```

Adds one behaviour label to the head of the list of behaviours associated with the action type. Note that just a label can be added to this list and not a complete behaviour definition. This can be done by adding a behaviour definition to the list of behaviours in the module and adding the label to the list of behaviour labels associated with the action. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int AppendBehaviour(ModeFE_Label *);
```

Adds one behaviour label to the end of the list of behaviours associated with the action type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int RemoveBehaviour(ModeFE_Label *);
```

Removes the first occurence of a behaviour label in the list of behaviour labels associated with the action type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise. This method does not suppress the behaviour definition associated with the label since the behaviour definition could be referenced in any other specification.

int ReplaceBehaviour(ModeFE_Label *, ModeFE_Label*);

Replaces the first occurence of the first parameter in the list of behaviour labels associated with the action type with the label given as a second parameter. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

void ClearBehaviours();

Removes all behaviour labels from the list of behaviour labels associated with the action type.

ModeFE_MODE GetMode();

Returns the mode associated with the action type. Possible values are:

- `CONFIRMED_`
- `NOTCONFIRMED_`

int SetMode(ModeFE_MODE);

Assigns a new value to the mode associated with the action type. The possible values for the parameter are:

- `CONFIRMED_`
- `NOTCONFIRMED_`

The methods returns `OK` if the operation was successfull. It returns `ERROR` otherwise.

ModeFE_NameList* GetParameterList();

Returns the list of parameter labels associated with the action type. If no parameter is specified then the methods returns an empty list (see section on lists for details).

int AddParameter(ModeFE_Label*);

Adds one parameter label to the head of the list of parameters associated with the action type. The methods returns `OK` if the operation was successfull. It returns `ERROR` otherwise.

int AppendParameter(ModeFE_Label*);

Adds one parameter label to the end of the list of parameters associated with the action type. The methods returns `OK` if the operation was successfull. It returns `ERROR` otherwise.

int RemoveParameter(ModeFE_Label*);

Removes the first occurence of the parameter label given in parameter in the list of parameter labels associated with the action type. The methods returns `OK` if the operation was successfull. It returns `ERROR` otherwise.

int ReplaceParameter(ModeFE_Label*, ModeFE_Label *);

Replaces the first occurence of the parameter label given in the first parameter with the second parameter in the list of parameter labels associated with the action type. The methods returns `OK` if the operation was successfull. It returns `ERROR` otherwise.

void ClearParameters();

Removes all parameter labels from the list of parameters associated with the action type.

```
ModeFE_Type GetInformationSyntax();
```

Returns the information syntax of the action type. If no information syntax was specified for the action type, the method returns `NULL`.

```
int SetInformationSyntax(ModeFE_Type);
```

Assigns a new information syntax to the action type. This value can be either a character string or `NULL` if no syntax is defined.

```
ModeFE_Type GetReplySyntax();
```

Returns the reply syntax of the action type . If no reply syntax was specified for the action type, the method returns `NULL`.

```
int SetReplySyntax(ModeFE_Type);
```

Assigns a new reply syntax to the action type. This value can be either a character string or `NULL` if no syntax is defined.

```
ModeFE_Action* Dupplicate();
```

Returns a pointer to a copy of the current object. The invoker is then responsible for deleting this pointer and its content.

```
void Print( ofstream * );
```

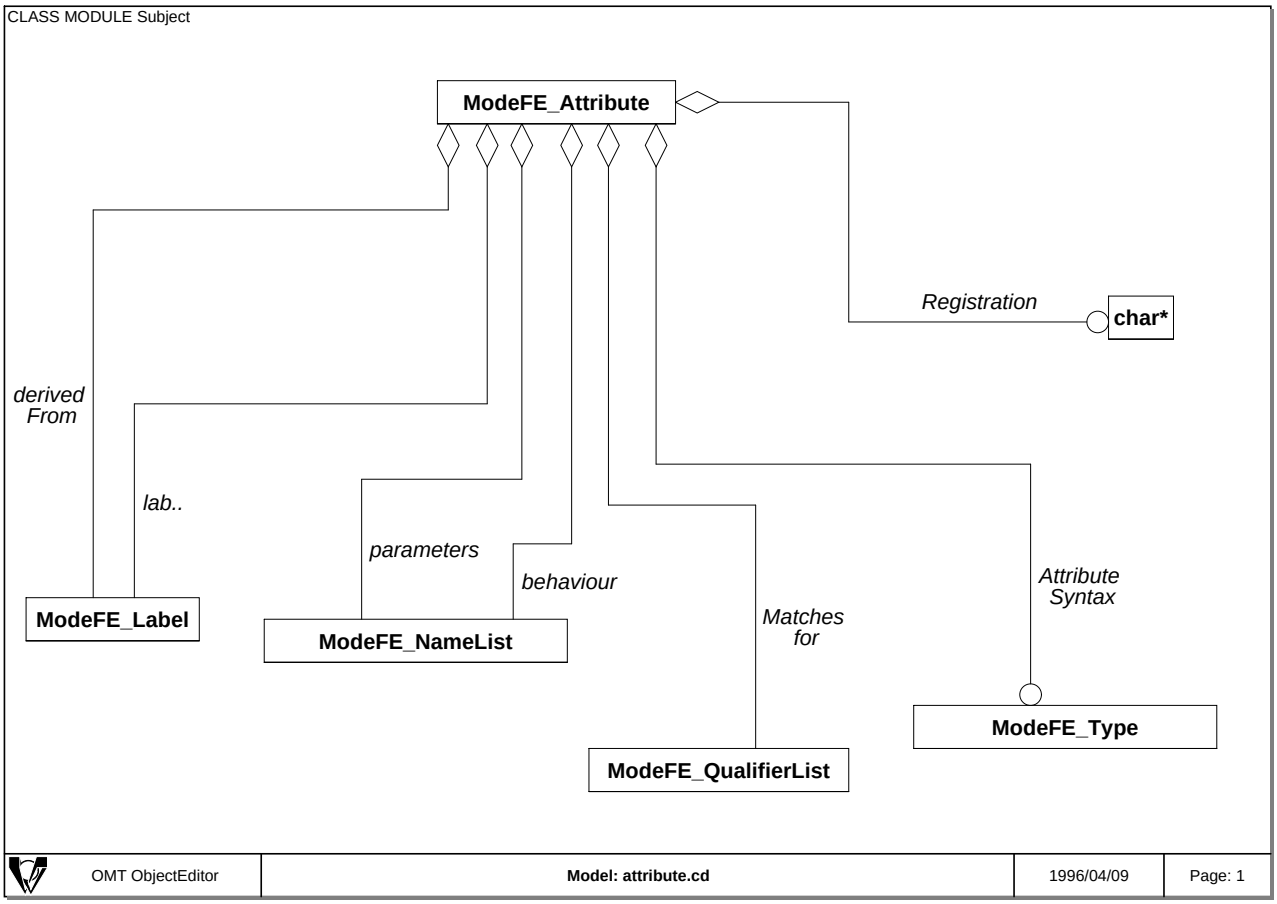
Prints in an ASCII format to the stream, the GDMO specification that corresponds tot this action type specification.

A.1.2 The ModeFE_Attribute Class

Purpose

The ModeFE_Attribute Class stores information related to the specification of a GDMO attribute type. An instance of that class contains all information related to one attribute specification from its label and registration identifier to its type, father attribute, parameters, behaviours and matching operations.

Architecture



Available methods

```
ModeFE_Attribute ( ModeFE_Label *,
                   char *,
                   ModeFE_Label *,
                   ModeFE_Type ,
                   ModeFE_QualifierList*,
                   ModeFE_NameList * ,
                   ModeFE_NameList * );
```

Basic constructor for the attribute class. The parameters are:

1. the attribute's type label,
2. the registration identifier (as a string without start and stop brackets),
3. the label of the attribute type from which this one is derived, (`NULL` if none),
4. the attribute's ASN.1 type (as a string or `NULL` if none),
5. the list of qualifiers associated with the attibute type (GET, GET-REPLACE, ...),
6. the list of behaviour labels associated with the attribute type,
7. the list of parameter labels associated with the attribute type.

```
ModeFE_Attribute ();
```

Default construtor for the ModeFE_Attribute class. Creates an empty instance of an attribute type.

```
ModeFE_Attribute (const ModeFE_Attribute&);
```

Copy constructor of the ModeFE_Attribute class. Returns no element in the current version of the library.

```
~ModeFE_Attribute ();
```

Destructor of the ModeFE_Attribute class. Deletes all contained elements.

```
ModeFE_Label* GetLabel();
```

Returns a pointer to a copy of the attribute type label. If no label is associated, returns a pointer to an empty ModeFE_Label object.

```
void SetLabel(ModeFE_Label* );
```

Assigns a new value to the attribute type label.

```
char* GetRegistration();
```

Returns a pointer to a copy of the attribute type registration identifier. The identifier is provided as a string and does not contain the "{" and "}" brackets.

```
void SetRegistration(char *);
```

Assigns a new registration identifier to the attribute specification. The registration identifier should not include the start and stop brackets ("{","}").

```
ModeFE_Label* GetDerivedFrom();
```

Returns the label of the attribute type from which this attribute type is derived. If this attribute is not derived from another one, then the result is an empty label (see ModeFE_Label definition for the specification of an empty label).

```
void SetDerivedFrom(ModeFE_Label* );
```

Assigns a new father attribute type to the current attribute type. If the parameter is an empty label or has the value `NULL`, the value stored in this attribute type will be an empty label meaning that this attribute does not derive from another one.

```
ModeFE_Type GetType();
```

Returns the ASN.1 type associated with this attribute type. If the value is `NULL`, then no type is associated with the attribute which is then derived from another one.

```
void SetType(ModeFE_Type );
```

Assigns a new ASN.1 type to the attribute specification. If the parameter is equal to `NULL`then no ASN.1 type is considered being associated with the attribute specification.

```
ModeFE_QualifierList* GetMatches();
```

Returns the list of all qualifiers associated with the attribute type. If no qualifier is associated, then the methods returns an empty qualifiers list.

int AddQualifier(ModeFE_QUALIFIERS);

Adds one qualifier to the head of the list of qualifiers associated with the attribute. Possible values for the parameter are:

- `NoQualifier_`,
- `EQUALITY_`,
- `ORDERING_`,
- `SUBSTRINGS_`,
- `SET_COMPARISON_`,
- `SET_INTERSECTION_`.

This method returns `OK` if the operation was successful, `FALSE` otherwise.

int AppendQualifier(ModeFE_QUALIFIERS);

Adds one qualifier to the end of the list of qualifiers associated with the attribute. Possible values for the parameter are:

- `NoQualifier_`,
- `EQUALITY_`,
- `ORDERING_`,
- `SUBSTRINGS_`,
- `SET_COMPARISON_`,
- `SET_INTERSECTION_`.

This method returns `OK` if the operation was successful, `FALSE` otherwise.

int RemoveQualifier(ModeFE_QUALIFIERS);

Removes the first occurence of the qualifier given in the parameter from the list of qualifiers associated with the attribute. Possible values for the parameter are:

- `NoQualifier_`,
- `EQUALITY_`,
- `ORDERING_`,
- `SUBSTRINGS_`,
- `SET_COMPARISON_`,
- `SET_INTERSECTION_`.

This method returns `OK` if the operation was successful, `FALSE` otherwise.

int ReplaceQualifier(ModeFE_QUALIFIERS, ModeFE_QUALIFIERS);

Replaces the first occurence of the first parameter with the second one in the list of qualifiers associated with the attribute. Possible values for the parameters are:

- `NoQualifier_`,

- EQUALITY_ ,
- ORDERING_ ,
- SUBSTRINGS_ ,
- SET_COMPARISON_
- SET_INTERSECTION_

This method returns OK if the operation was successful, FALSE otherwise.

void ClearQualifier();

Removes all qualifiers associated with the attribute type.

ModeFE_NameList* GetBehaviours();

Returns a pointer to a copy of the list of all behaviour labels assiciated with the attribute type. If no behaviour is associated, the method returns a emtpy list.

int AddBehaviour(ModeFE_Label *);

Adds one behaviour label to the head of the list of behaviours associated with the action type. The method returns OK if the operation was successful. It returns ERROR otherwise.

int AppendBehaviour(ModeFE_Label *);

Adds one behaviour label to the end of the list of behaviours associated with the attribute type. This method returns OK if the operation was successful. It returns ERROR otherwise.

int RemoveBehaviour(ModeFE_Label *);

Removes the first occurence of a behaviour label in the list of behaviour labels associated with the attribute type. This method returns OK if the operation was successful. It returns ERROR otherwise.

int ReplaceBehaviour(ModeFE_Label *, ModeFE_Label *);

Replaces the first occurence of the first parameter in the list of behaviour labels associated with the attribute type with the label given as a second parameter. This method returns OK if the operation was successful. It returns ERROR otherwise.

void ClearBehaviours();

Removes all behaviour labels from the list of behaviour labels associated with the attribute type.

ModeFE_NameList* GetParameters();

Returns the list of parameter labels associated with the attribute type. If no parameter is specified then the methods returns an empty list.

int AddParameter(ModeFE_Label *);

Adds one parameter label to the head of the list of parameters associated with the attribute type. The methods returns OK if the operation was successfull. It returns ERROR otherwise.

int AppendParameter(ModeFE_Label *);

Adds one parameter label to the end of the list of parameters associated with the attribute type. The methods returns `OK` if the operation was successfull. It returns `ERROR` otherwise.

int RemoveParameter(ModeFE_Label *);

Removes the first occurence of the parameter label given in parameter in the list of parameter labels associated with the attribute type. The methods returns `OK` if the operation was successfull. It returns `ERROR` otherwise.

int ReplaceParameter(ModeFE_Label *, ModeFE_Label *);

Replaces the first occurence of the parameter label given in the first parameter with the second parameter in the list of parameter labels associated with the attribute type. The methods returns `OK` if the operation was successfull. It returns `ERROR` otherwise.

void ClearParameters();

Removes all parameter labels from the list of parameters associated with the attribute type.

ModeFE_Attribute* Dupplicate();

Returns a pointer to a copy of the current attribute object.

void Print(ofstream *) ;

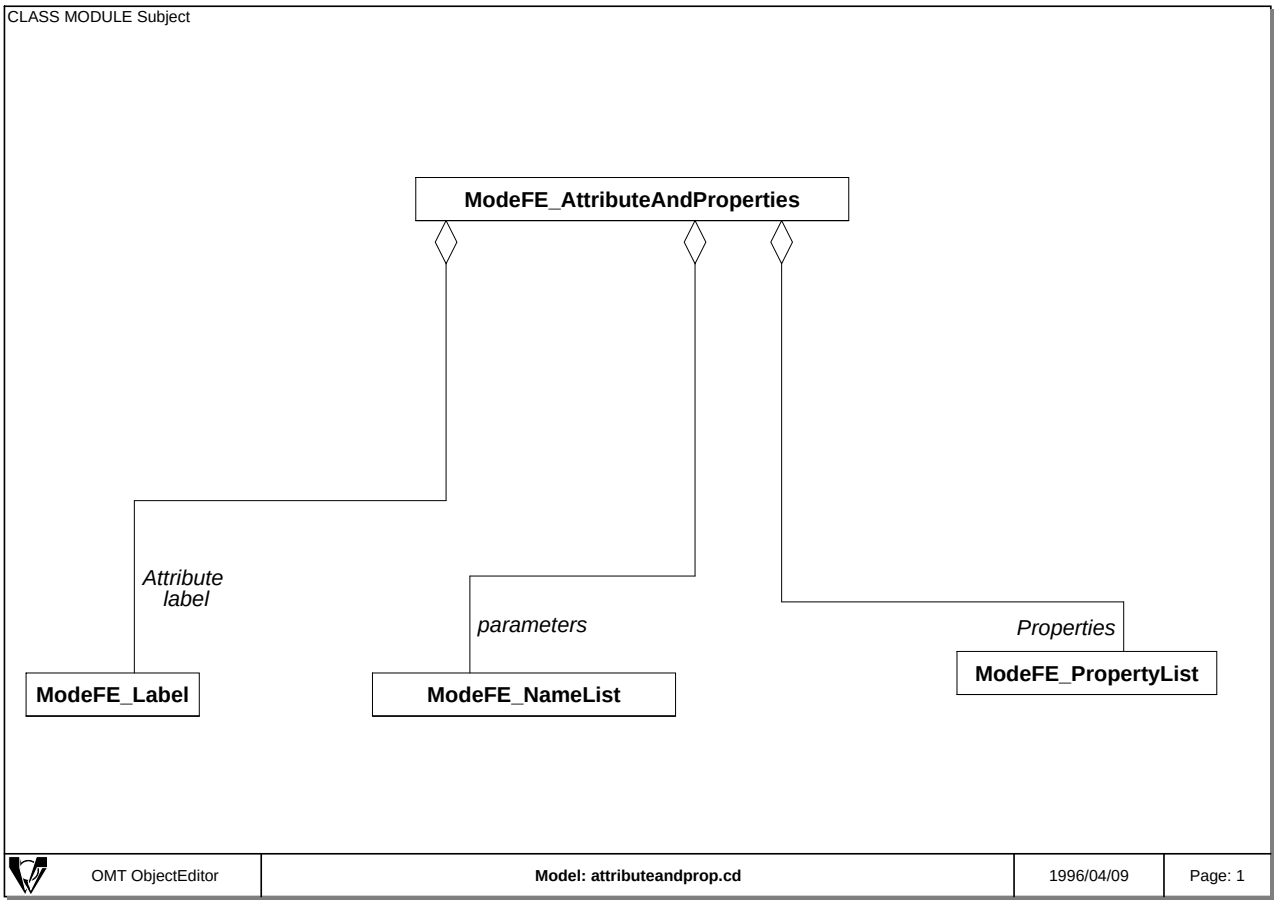
Prints in an ASCII format to the stream, the GDMO specification that corresponds tot this attribute type specification.

A.1.3 The ModeFE_AttributeAndProperties class

Purpose

The ModeFE_AttributeAndPorperties class is used to represent an attribute and its properties as they are defined in a package type. It thus contains an attribute label, a list of associated propeties and a list of associated parameters.

Architecture



Available methods

```
ModeFE_AttributeAndProperties (ModeFE_Label *,
                               ModeFE_PropertyList *,
                               ModeFE_NameList *);
```

```
ModeFE_AttributeAndProperties ();
```

Default constructor for the ModeFE_AttributeAndProperties. It provides an empty instance of that object.

```
ModeFE_AttributeAndProperties (const ModeFE_AttributeAndProperties&);
```

Copy constructor of the ModeFE_AttributeAndProperties class. Returns no element in the current version of the library.

```
~ModeFE_AttributeAndProperties();
```

Destructor of the ModeFE_AttributeAndPropeties class. Deletes all contained elements.

```
ModeFE_Label* GetLabel();
```

Returns a pointer to a copy of the label of the attribute type concerned. If no label is associated, returns a pointer to an empty ModeFE_Label object.

int SetLabel(ModeFE_Label*);

Assigns a new value to the label that specifies which attribute is concerned. Returns `OK` if the operation was successful. Returns `ERROR` otherwise.

ModeFE_PropertyList* GetProperties();

Returns the list of properties associated with the attribute. If no property exists, the method returns an empty property list.

int AddProperty(ModeFE_OneProperty *);

Adds one property to the head of the list of properties associated with the attribute. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AppendProperty(ModeFE_OneProperty *);

Adds one property to the end of the list of properties associated with the attribute. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int RemoveProperty(ModeFE_OneProperty *);

Removes the first occurence of the property from the list of properties associated with the attribute. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise. The selection crieria is based on the property type (see the ModeFE_OneProperty class) for the list of possible property types).

int ReplaceProperty(ModeFE_OneProperty *, ModeFE_OneProperty*);

Replaces the first occurence of the first parameter with the second in the list of properties associated with the attribute. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise. The selection crieria is based on the property type (see the ModeFE_OneProperty class) for the list of possible property types).

void ClearProperties();

Removes all properties associated with the attribute.

ModeFE_NameList* GetParameters();

Returns the list of parameter labels associated with the attribute type in the package. If no parameter is specified then the methods returns an empty list (see section on lists for details).

int AddParameter(ModeFE_Label *);

Adds one parameter label to the head of the list of parameters associated with the attribute type in a pckage. The methods returns `OK` if the operation was successfull. It returns `ERROR` otherwise.

int AppendParameter(ModeFE_Label *);

Adds one parameter label to the end of the list of parameters associated with the attribute type in a pckage. The methods returns `OK` if the operation was successfull. It returns `ERROR` otherwise.

int RemoveParameter(ModeFE_Label *);

Removes the first occurence of the parameter label given in parameter in the list of parameter labels associated with the attribute type in the package. The methods returns `OK` if the operation was successfull. It returns `ERROR` otherwise.

int ReplaceParameter(ModeFE_Label *);

Replaces the first occurence of the parameter label given in the first parameter with the second parameter in the list of parameter labels associated with the attribute type in the package. The methods returns `OK` if the operation was successfull. It returns `ERROR` otherwise.

void ClearParameters();

Removes all parameter labels from the list of parameters associated with the attribute type in the package.

ModeFE_AttributeAndProperties* Dupplicate();

Returns a pointer to a copy of the current object. The invoker is then responsible for deleting this pointer and its content.

void Print (ofstream *) ;

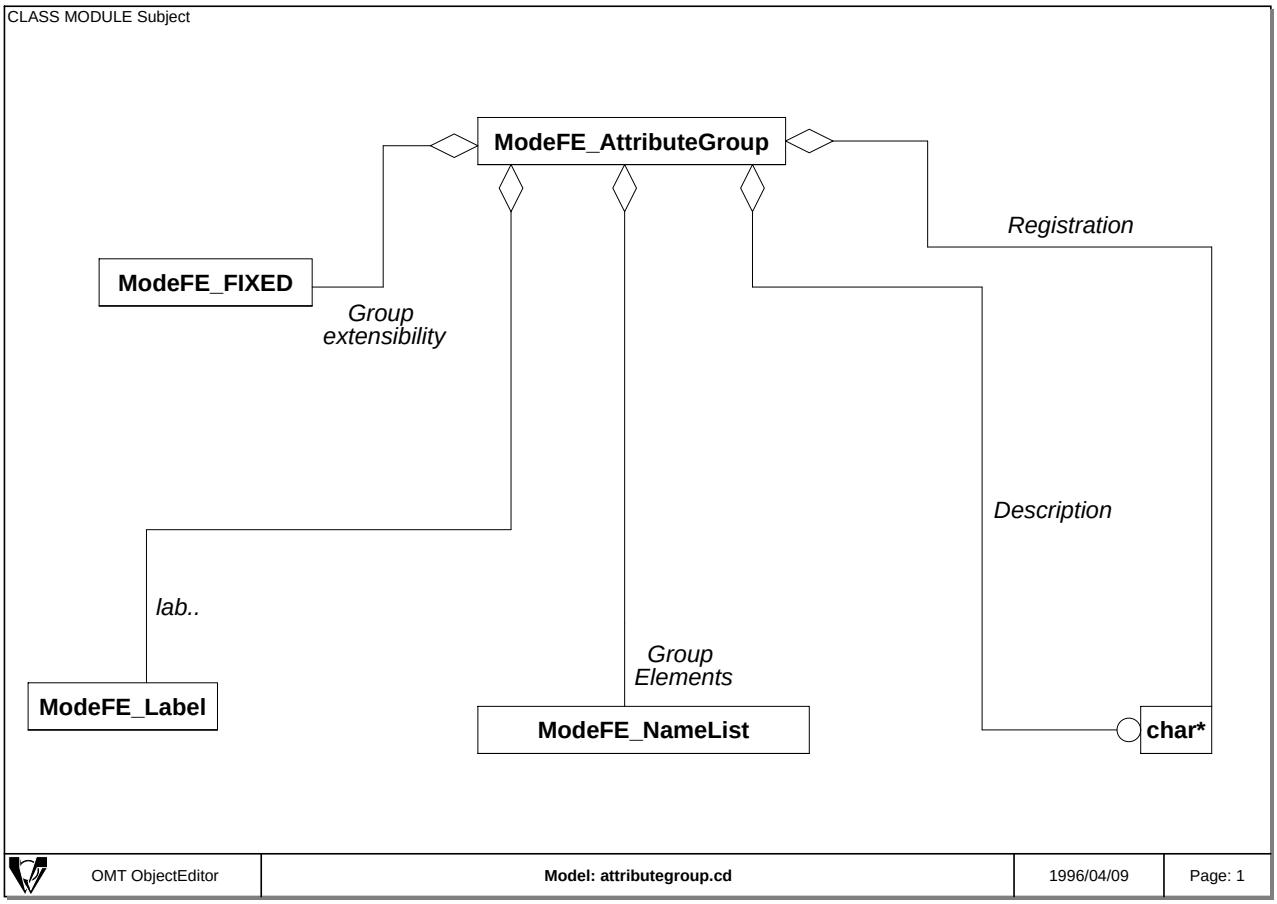
Prints in an ASCII format to the stream, the GDMO specification that corresponds tot this action type specification.

A.1.4 The ModeFE_AttributeGroup class

Purpose

The ModeFE_AttributeGroup class stores information related to the specification of a GDMO Attribute group type. It contains the attribute type label, the registration identifier, the labels of attribute types that are within the group, a statement on the extensibility of the group and its informal description.

Architecture



Available methods

```
ModeFE_AttributeGroup ( ModeFE_Label *,
                        char * ,
                        ModeFE_NameList * ,
                        ModeFE_FIXED ,
                        char * );
```

Basic constructor for an attribute group object. The parameters are:

1. the label associated with the attribute group type,
2. the registration identifier of the definition without the "{" and "}" brackets,
3. the list of the attribute labels whose attributes are grouped into this attribute group,
4. a statement on whether the group is fixed or not. Poosible values are:
   - `FIXED_`
   - `NOTFIXED_`
5. a textual description of the group.

```
ModeFE_AttributeGroup ();
```

Constructor for the ModeFE_AttributeGroup class which creates an empty ModeFE_AttributeGroup definition.

ModeFE_AttributeGroup (**const** ModeFE_AttributeGroup&);

Copy constructor of the ModeFE_AttributeGroup class. Returns no element in the current version of the library.

~ModeFE_AttributeGroup ();

Destructor of the ModeFE_AttributeGroup class. Deletes all contained elements.

ModeFE_Label* GetLabel();

Returns a pointer to a copy of the attribute group type label. If no label is associated, returns a pointer to an empty ModeFE_Label object.

int SetLabel(ModeFE_Label*);

Assigns a new value to the attribute group type label. Returns `OK` if the operation was successful. Returns `ERROR` otherwise.

char* GetRegistration();

Returns a pointer to a copy of the attribtue group type registration identifier. The identifier is provided as a string and does not contain the "{" and "}" brackets.

void SetRegistration(**char***);

Assigns a new registration identifier to the ttribute group specification. The registration identifier should not include the start and stop brackets ("{","}").

ModeFE_NameList* GetGroupElements();

Returns a pointer to a copy of the list of attribute labels grouped within this attribute group. IF no attribute is defined in the group, the method returns an empty list.

int AddGroupElement(ModeFE_Label *);

Adds one attribute label to the head of the list of attributes grouped within this attribute group. The methods returns `OK` if the operation was successful, `ERROR`otherwise.

int AppendGroupElement(ModeFE_Label *);

Adds one attribute label to the end of the list of attributes grouped within this attribute group. The methods returns `OK` if the operation was successful, `ERROR`otherwise.

int RemoveGroupElement(ModeFE_Label *);

Removes one attribute label from the list of attribute labels grouped in the attribute group. The methods returns `OK` if the operation was successful, `ERROR`otherwise.

int ReplaceGroupElement(ModeFE_Label *, ModeFE_Label*);

Replaces the first label given in parameter with the second in the list of attribute labels gruped in the attribute group. The methods returns `OK` if the operation was successful, `ERROR`otherwise.

void ClearGroupElements();

Removes all attribute labels from the list of attribute labels grouped in the attribute group.

ModeFE_FIXED GetFixed();

Returns the extensibility status of the group. Possible values are:

- `FIXED_`
- `NOTFIXED_`

int SetFixed(ModeFE_FIXED);

Assigns a new extensibility status to the attribute group. Possible values for the parameter are:

- `FIXED_`
- `NOTFIXED_`

char* GetDescription();

Returns a pointer to a copy of the attribute grup description. If no description exists, the methods returns `NULL`.

int SetDescription(**char***);

Assigns a new description to the attribute group.

ModeFE_AttributeGroup* Dupplicate();

Returns a pointer to a copy of the attribute group.

void Print(ofstream *) ;

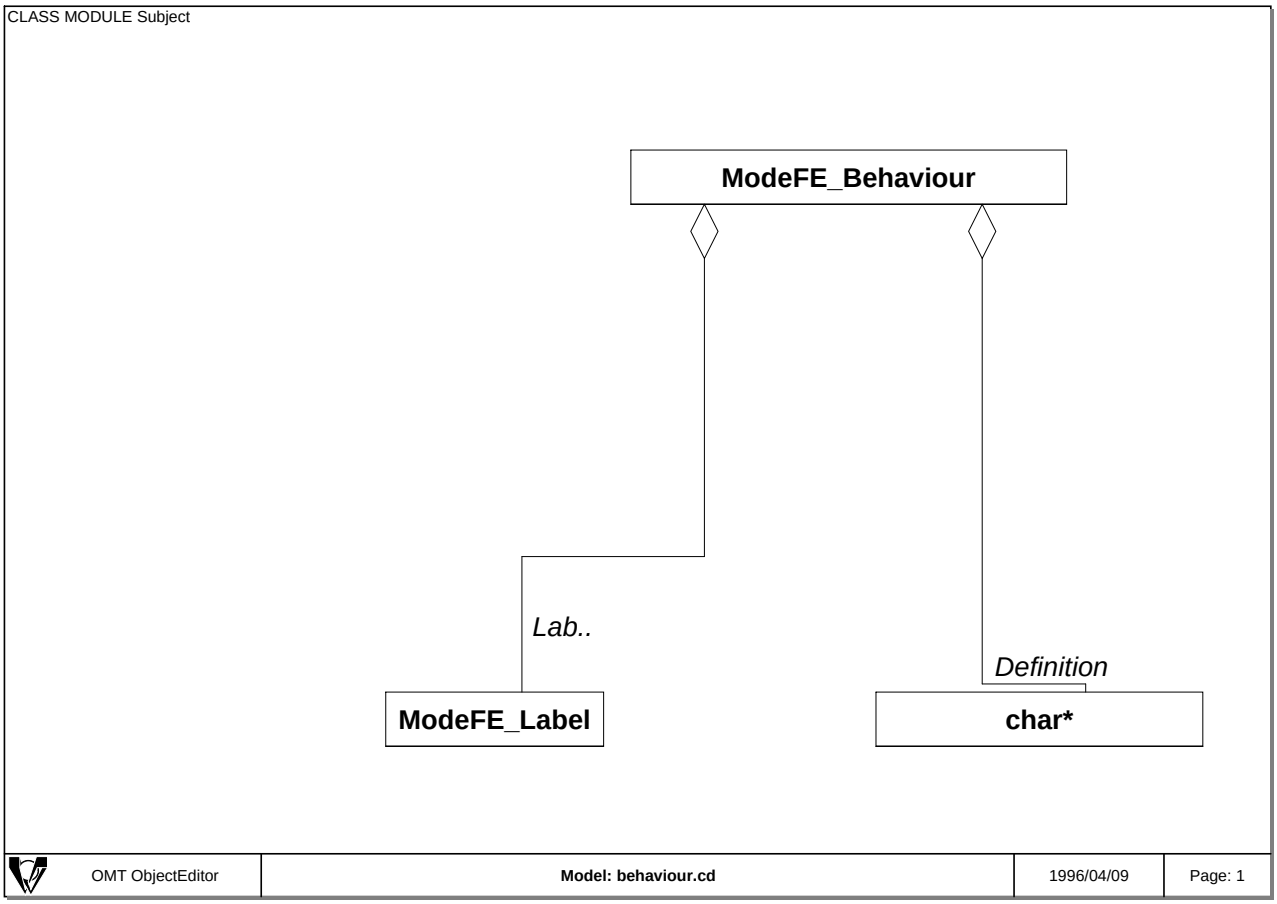
Prints in an ASCII format to the stream, the GDMO specification that corresponds tot this attribute group type specification.

A.1.5 The ModeFE_Behaviour class

Purpose

The ModeFE_Behaviour Class stores information related to the specification of a GDMO Behaviour template. It contains the behaviour label, qnd the qssociqted informal specification.

Architecture



Available methods

```
ModeFE_Behaviour (ModeFE_Label *,
                  char * b);
```

Basic constructor for a behaviour object. The parameters are:

1. the label associated with the behaviour specification,
2. its definition.

```
ModeFE_Behaviour ();
```

Constructor for the ModeFE_Behaviour class which creates an empty ModeFE_Behaviour instance.

```
ModeFE_Behaviour (const ModeFE_Behaviour&);
```

Copy constructor of the ModeFE_Behaviour class. Returns no element in the current version of the library.

```
~ModeFE_Behaviour ();
```

Destructor of the ModeFE_behaviour class. Deletes all contained elements.

```
ModeFE_Label * GetLabel();
```

Returns a pointer to a copy of the behaviour type label. If no label is associated, returns a pointer to an empty ModeFE_Label object.

int SetLabel(ModeFE_Label *);

Assigns a new value to the behaviour type label. Returns `OK` if the operation was successful. Returns `ERROR` otherwise.

char * GetDefinition();

Returns a pointer to a copy of the behaviours definition. If no definition is present, the methods returns `NULL`.

int SetDefinition(**char** *);

Assigns a new definition to the behaviour type.

ModeFE_Behaviour* Dupplicate();

Returns a pointer to a copy of the behaviour type.

void Print(ofstream *) ;

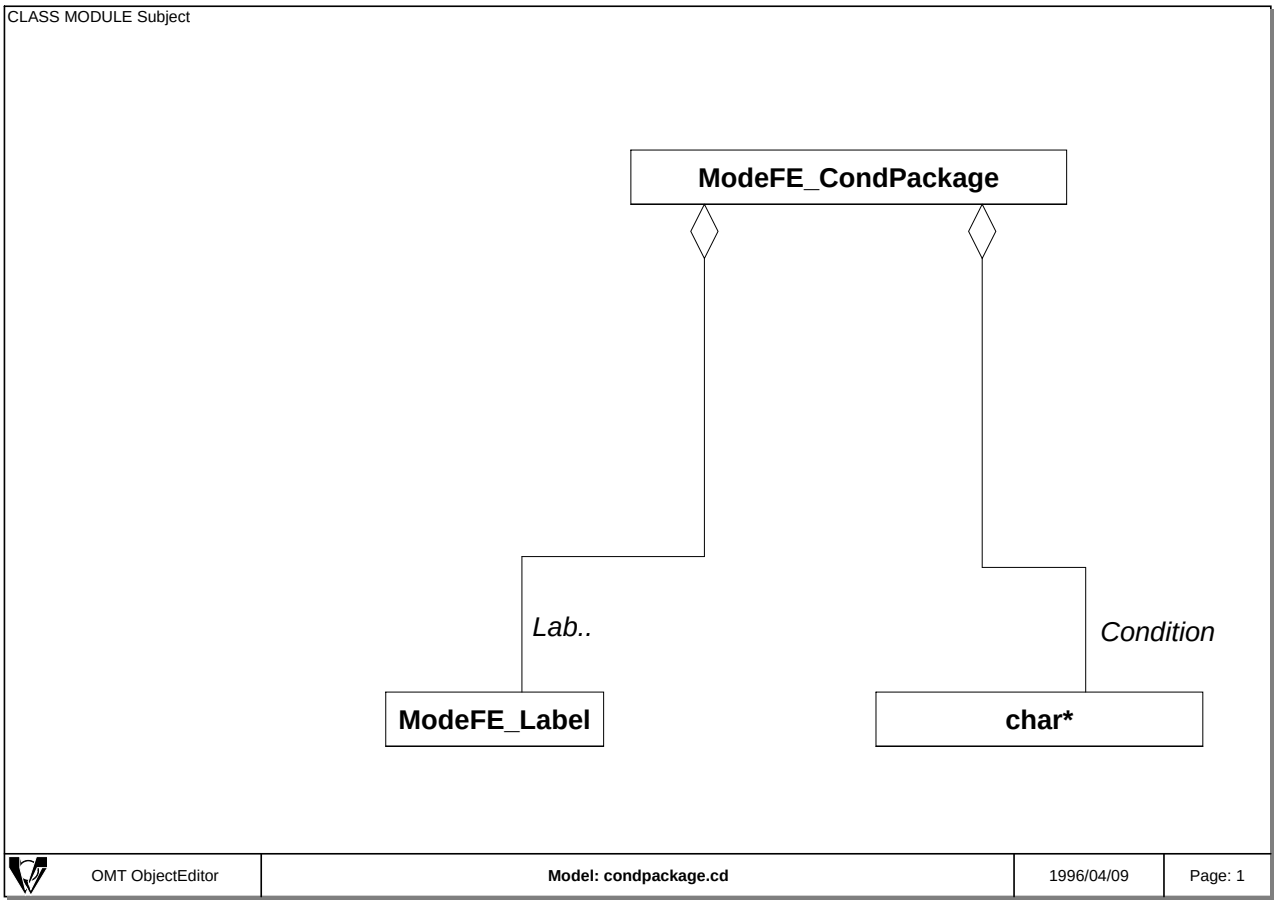
Prints in an ASCII format to the stream, the GDMO specification that corresponds tot this behaviour type.

A.1.6 The ModeFE_CondPackage class

Purpose

The ModeFE_CondPackage Class stores information related to the specification of a conditional packaged referenced within one or more managed object classes. This class contains information on the label of the package that is declared as conditional and the definition of the condition that defines its presence in a managed object instance.

Architecture



Available methods

ModeFE_CondPackage (ModeFE_Label *, **char** *);

Basic constructor for the ModeFE_CondPackage class. The parameters are:

1. the label of the package which is conditional,
2. the condition defined in an informal way.

ModeFE_CondPackage ();

Constructor for the ModeFE_CondPackage class which creates an empty ModeFE_CondPackage definition.

ModeFE_CondPackage(**const** ModeFE_CondPackage&);

Copy constructor of the ModeFE_CondPackage class. Returns no element in the current version of the library.

~ModeFE_CondPackage ();

Destructor of the ModeFE_CondPackage class. Deletes all contained elements.

ModeFE_Label* GetLabel();

Returns a pointer to a copy of the conditional package type label. If no label is associated, the method returns a pointer to an empty ModeFE_Label object.

void SetLabel(ModeFE_Label*);

Assigns a new value to the conditional package type label.

char* GetCondition();

Returns a pointer to a copy of the condition which states under which circumstances the package is present in a managed object instance.

void SetCondition(**char***);

Assigns a new presence condition to the conditional package.

int IsEqual(ModeFE_CondPackage*);

Returns `TRUE` if the conditional parameter given in parameter is equal to the current one. Returns `FALSE` otherwise. The equality test is made only on the package label and not on the condition.

ModeFE_CondPackage* Dupplicate();

Returns a pointer to a copy of the current object. The invoker is then responsible for deleting this pointer and its content.

void Print(ofstream *) ;

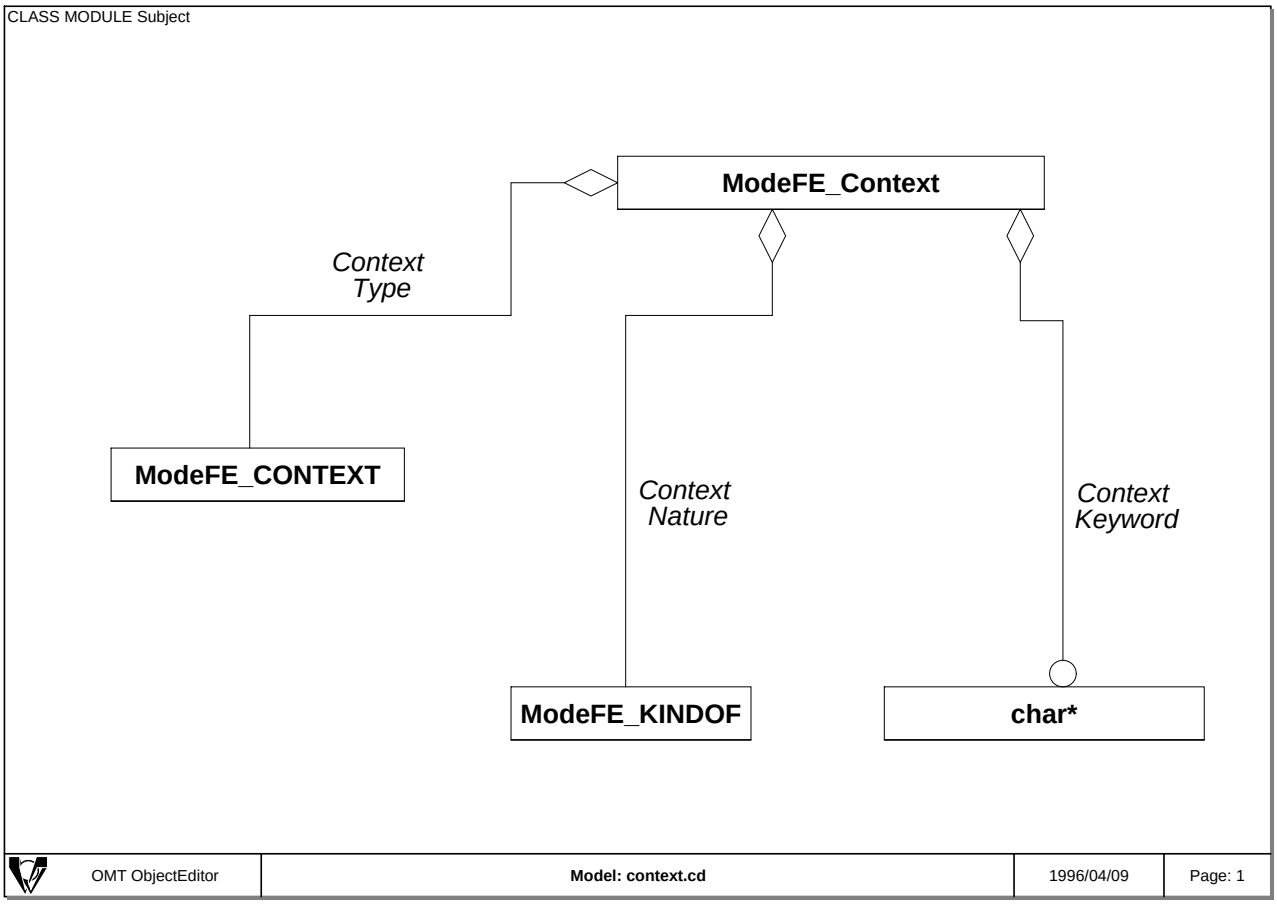
Prints in an ASCII format to the stream, the GDMO specification that corresponds tot this conditional package type.

A.1.7 The ModeFE_Context class

Purpose

The ModeFE_Context Class stores information related to the usage context of a parameter. This class is used in a parameter class and contains information on the kind of context defined for the parameter (qualifier or keyword), the context type and the associated context keyword if any.

Architecture



Available methods

ModeFE_Context (**char** * a);

Basic constructor for the ModeFE_Context class. Assigns the value of the parameter to the context keywork and defines the context kind as `KEYWORD_`

ModeFE_Context (ModeFE_CONTEXT a);

Basic constructor for the ModeFE_Context class. Assigns the value of the parameter to the context type and defines the context kind as `QUALIFIER_`. In this case no context keyword exists for the context. Possible values for the parameter are:

- `NoContext_`
- `ACTION_INFO_`
- `ACTION_REPLY_`
- `EVENT_INFO_`
- `EVENT_REPLY_`
- `SPECIFIC_ERROR_`

ModeFE_Context ();

Default constructor for the ModeFE_Context class.

```
ModeFE_Context (const ModeFE_Context&);
```

Copy constructor of the ModeFE_Context class. Returns no element in the current version of the library.

```
~ModeFE_Context ();
```

Destructor of the ModeFE_Context class. Deletes all contained elements.

```
ModeFE_KINDOF GetWhich();
```

Returns the kind of context instanciated within the parameter specification. Possible values are:

- `QUALIFIER_`: the context is defined by a qualifier,
- `KEYWORD_`: the context is defined by a context keyword.

```
int SetWhich(ModeFE_KINDOF);
```

Assigns a new semantics to the context by defining either a qualifier based context or a keyword based context. Possible values for the method parameter are:

- `QUALIFIER_`: the context is defined by a qualifier,
- `KEYWORD_`: the context is defined by a context keyword.

```
char* GetKeyword();
```

Returns the context keyword. If no context keyword exists, the method returns `NULL`.

```
int SetKeyword(char*);
```

Assigns a new keyword to the context. The method returns `OK` if the operation was successful, `ERROR` otherwise.

```
ModeFE_CONTEXT GetContext();
```

Returns the context qualifier type defined in this context. This value is only to be considered if the context kind is equal to `QUALIFIER_` Possible values for the context qualifier type are:

- `NoContext_`
- `ACTION_INFO_`
- `ACTION_REPLY_`
- `EVENT_INFO_`
- `EVENT_REPLY_`
- `SPECIFIC_ERROR_`

```
int SetContext(ModeFE_CONTEXT);
```

Assigns a new qualifier value to the context; Possible values for the method parameter are:

- `NoContext_`

- ACTION_INFO_
- ACTION_REPLY_
- EVENT_INFO_
- EVENT_REPLY_
- SPECIFIC_ERROR_

The method returns OK if the operation was successful, ERROR otherwise.

ModeFE_Context* Dupplicate();

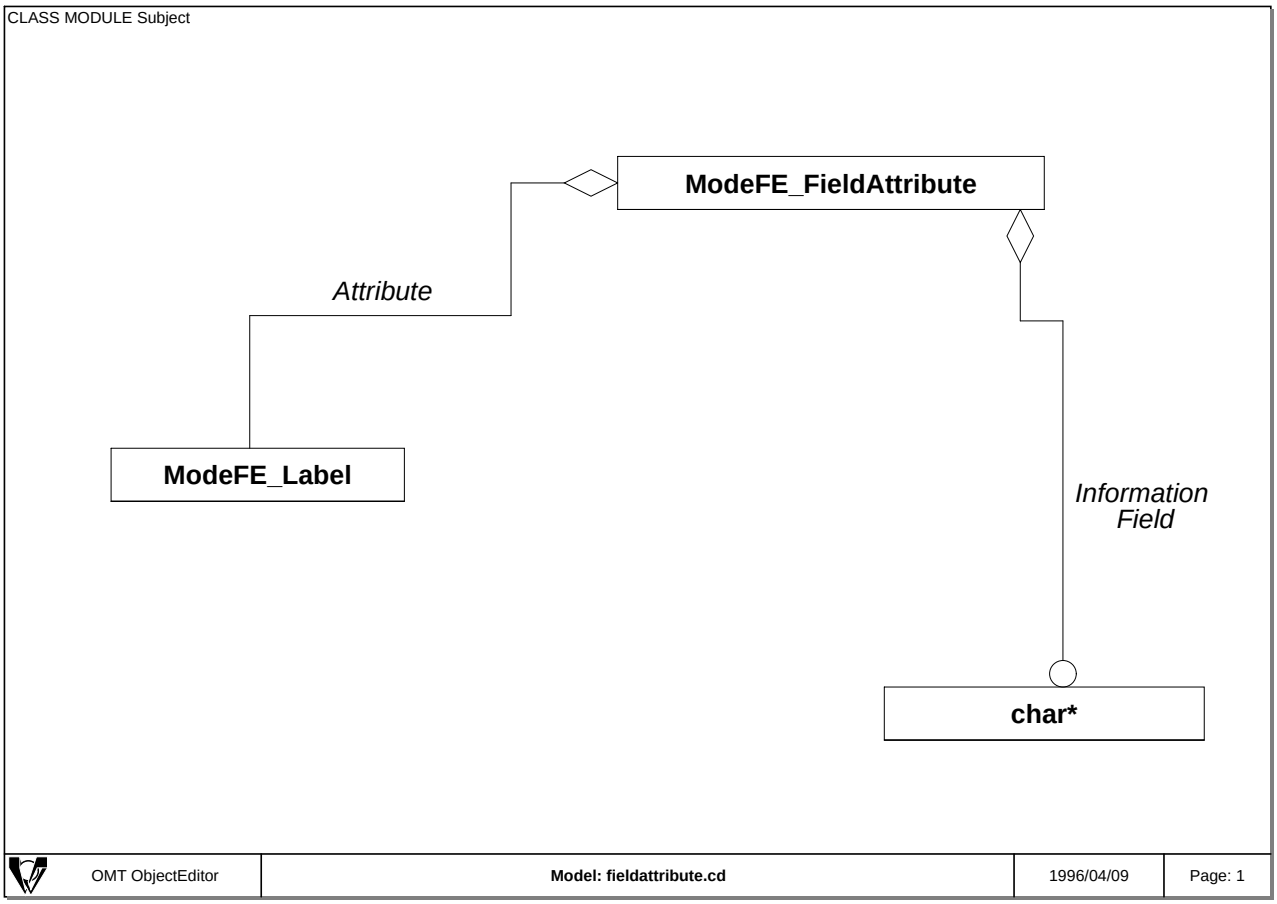
Returns a pointer to a copy of the current object. The invoker is then responsible for deleting this pointer and its content.

A.1.8 The ModeFE_FieldAttribute class

Purpose

The ModeFE_FieldAttribute Class stores information related to the association of an attribute label and a information syntax field within a notification invocation or an action reply. Used within a notification specification it stores an attribute label and the name of the field used in the notification for carrying the attribute value.

Architecture



Available methods

```
ModeFE_FieldAttribute ( char *,
                        ModeFE_Label *);
```

Basic constructor for the ModeFE_FieldAttribute class. The parameters of the constructor are:

1. the name of the field carrying the attribute value,
2. the label of the attribute associated with the field.

```
ModeFE_FieldAttribute ();
```

Default constructor of the ModeFE_FieldAttribute class. The constructor creates an empty instance with the field value assigned to `NULL` and the attribute having an empty attribute label.

```
ModeFE_FieldAttribute (const ModeFE_FieldAttribute&);
```

Copy constructor of the ModeFE_FieldAttribute class. Returns no element in the current version of the library.

```
~ModeFE_FieldAttribute ();
```

Destructor of the ModeFE_FieldAttribute class. Deletes all contained elements.

```
char* GetField();
```

Returns a pointer to a copy of the field name.

```
int SetField(char*);
```

Assigns a new field name which will be associated with a label. The method returns `OK` if the operation was successful, `ERROR` otherwise.

```
ModeFE_Label* GetAttribute();
```

Returns a pointer to a copy of the label of the attribute associated with the field. If no attribute is associated, the method returns an empty label.

```
void SetAttribute(ModeFE_Label *);
```

Assigns a new attribute label to the association such that the field will carry the value of the new associated attribute.

```
int IsEqual(ModeFE_FieldAttribute*);
```

Returns `TRUE` if the object given in parameter is equal to the current object. The equality test is only performed on the bases of the field name in this version of the library.

```
ModeFE_FieldAttribute* Dupplicate();
```

Returns a pointer to a copy of the current object. The invoker is then responsible for deleting this pointer and its content.

```
void Print( ofstream * ) ;
```

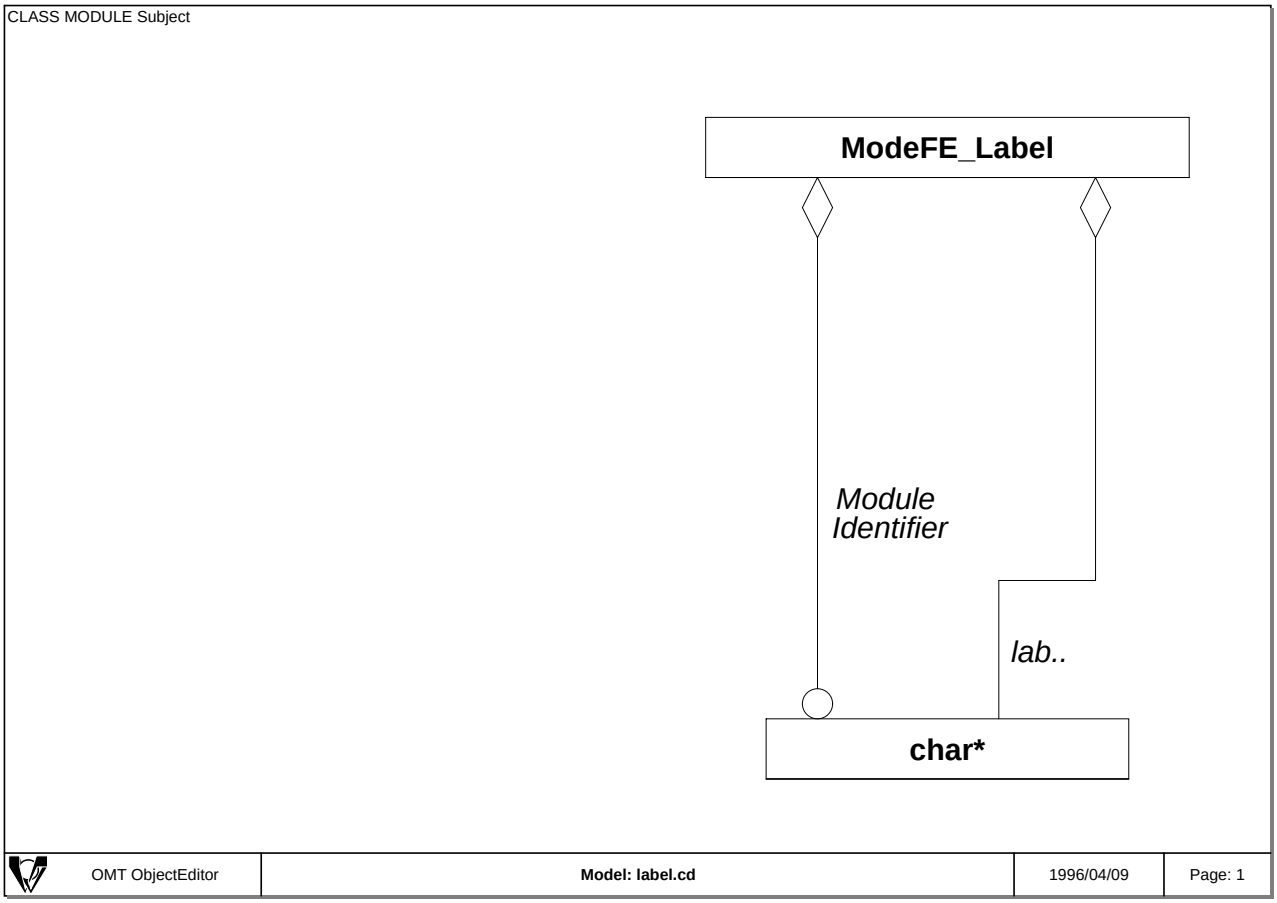
Prints in an ASCII format to the stream, the GDMO specification that corresponds tot this action type specification.

A.1.9 The ModeFE_Label class

Purpose

The ModeFE_Label Class stores information related to the identification label of any GDMO and GRM specification. An identification label contains an optional module identifier and a local label defined as a character string.

Architecture



Available methods

ModeFE_Label (**char** $*$, **char** $*$);

Basic constructor of the ModeFE_Label class. The parameters are:

1. the local label name,
2. the module identifier, if any `NULL` otherwise.

ModeFE_Label ();

Default constructor of the ModeFE_Label class. Builds an empty label, i.e. a specification label with an empty label (`NULL`) and an empty module identifier (`NULL`).

```
ModeFE_Label (const ModeFE_Label&);
```

Copy constructor of the ModeFE_Label class. Returns no element in the current version of the library.

```
~ModeFE_Label();
```

Destructor of the ModeFE_Label class. Deletes all contained elements.

```
char* GetLabel();
```

Returns a pointer to a copy of the local label of the specification label

```
int SetLabel(char *);
```

Assigns a new local label to the specification label. The method returns `OK` if the operation was successfull, `ERROR` otherwise.

```
char* GetModuleIdentifier();
```

Returns a pointer to a copy of the labels module identifier, if any. if no module identifier is defined for this label, it returns `NULL`.

```
int SetModuleIdentifier(char *);
```

Assigns a new module identifier to the specification label. The method returns `OK` if the operation was successfull, `ERROR` otherwise.

```
int IsEqual(ModeFE_Label *);
```

Returns `TRUE` if the label given as a parameter to this method is equal to the current label. Returns `FALSE` otherwise. The equality test is made on both module identifier and local label.

```
ModeFE_Label* Dupplicate();
```

Returns a pointer to a copy of the current object. The invoker is then responsible for deleting this pointer and its content.

```
void Print ( ofstream *) ;
```

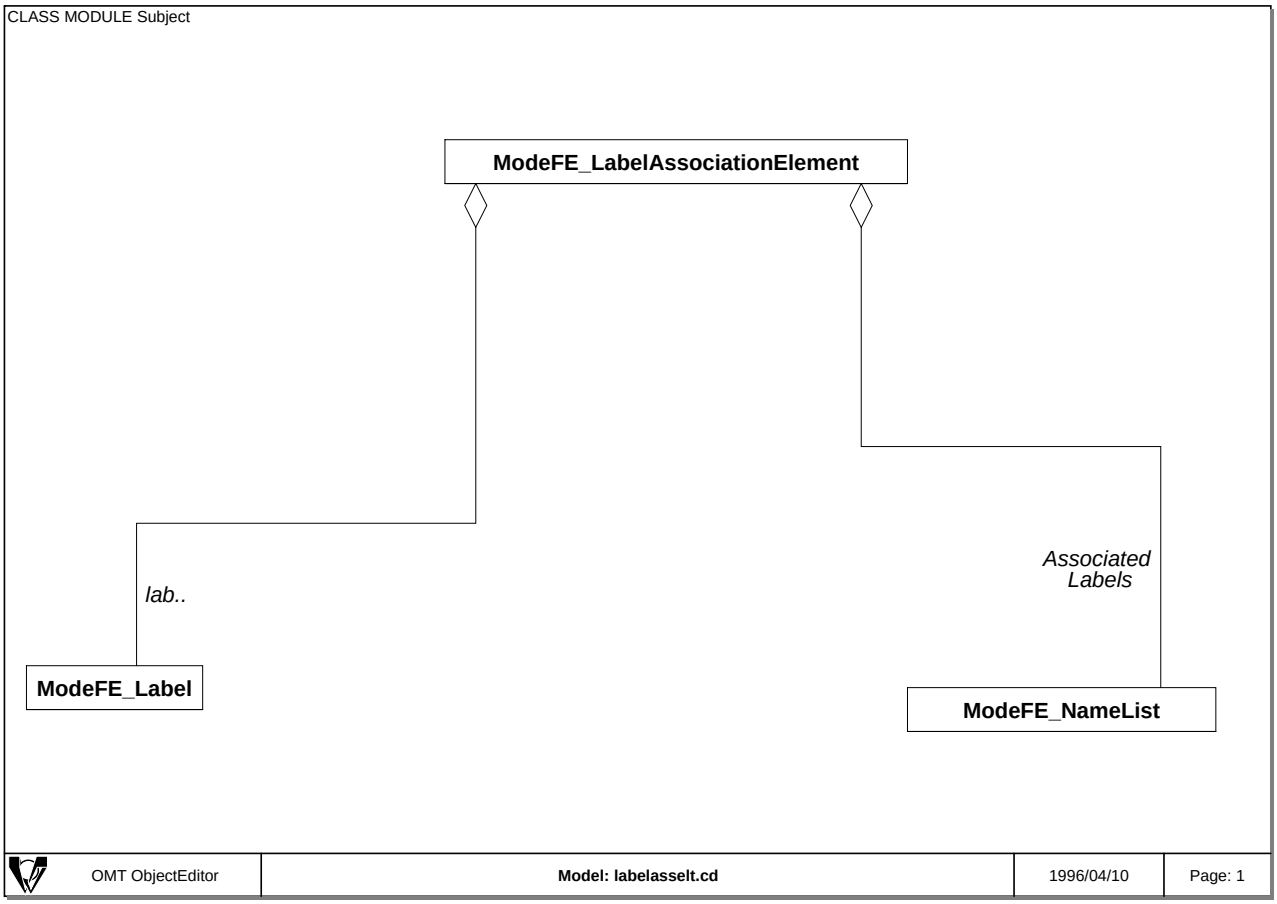
Prints in an ASCII format to the stream, the GDMO syntax that corresponds tot this label, i.e. "module-identifier":label.

A.1.10 The ModeFE_LabelAssociationElement class

Purpose

The ModeFE_LabelAssociationElement Class stores information related to the association of one label with a set of other ones. It is used within a package definition where parameters are associated with actions and notifications. This class allows to implement the one to many link between labels as required in this part of a package specification.

Architecture



Available methods

```
ModeFE_LabelAssociationElement (ModeFE_Label * ,
                                ModeFE_NameList*);
```

Basic constructor of the ModeFE_LabelAssociationElement. The parameters are:

1. the label which is associated with other one,
2. the list of associated labels.

```
ModeFE_LabelAssociationElement ();
```

Default constructor of the ModeFE_LabelAssociationElement. Creates an empty label and an empty association list.

```
ModeFE_LabelAssociationElement(const ModeFE_LabelAssociationElement&);
```

Copy constructor of the ModeFE_LabelAssociationElement class. Returns no element in the current version of the library.

```
~ModeFE_LabelAssociationElement();
```

Destructor of the ModeFE_LabelAssociationElement class. Deletes all contained elements.

```
ModeFE_Label* GetLabel();
```

Returns the label which is associated to a list of labels. If no label exists, the method returns an empty label.

```
int SetLabel(ModeFE_Label *);
```

Assigns a new label to the association. The method returns `OK` if the operation was successful, `ERROR` otherwise.

```
int IsEqual(ModeFE_LabelAssociationElement*);
```

Returns `TRUE` if the label association element given in parameter is equal to the current element. Returns `FALSE` otherwise. The equality test is made only on the label part of the association and not on the associate labels list.

```
ModeFE_NameList* GetAssociate();
```

Returns the list of labels associated with the association label. If no associate label exists, the method returns an empty list.

```
int AddAssociate (ModeFE_Label* );
```

Adds one label to the head of the list of associate labels. The method returns `OK` if the operation was successful, `ERROR` otherwise.

```
int AppendAssociate (ModeFE_Label* );
```

Adds one label to the end of the list of associate labels. The method returns `OK` if the operation was successful, `ERROR` otherwise.

```
int SetAssociate(ModeFE_NameList *);
```

Assigns the list of associate labels to the list given in parameter. The method returns `OK` if the operation was successful, `ERROR` otherwise.

```
int RemoveAssociate (ModeFE_Label* );
```

Removes the first occurence of the label given in parameter from the list of associate labels. The method returns `OK` if the operation was successful, `ERROR` otherwise.

```
int ReplaceAssociate (ModeFE_Label*, ModeFE_Label *);
```

Replaces the first occurence of the first parameter in the list of associate labels with the label given as a second parameter. This method returns `OK` if the operation was successful, `ERROR` otherwise.

void ClearAssociate();

Removes all associate labels from the association

ModeFE_LabelAssociationElement* Dupplicate();

Returns a pointer to a copy of the current object. The invoker is then responsible for deleting this pointer and its content.

void Print (ofstream *) ;

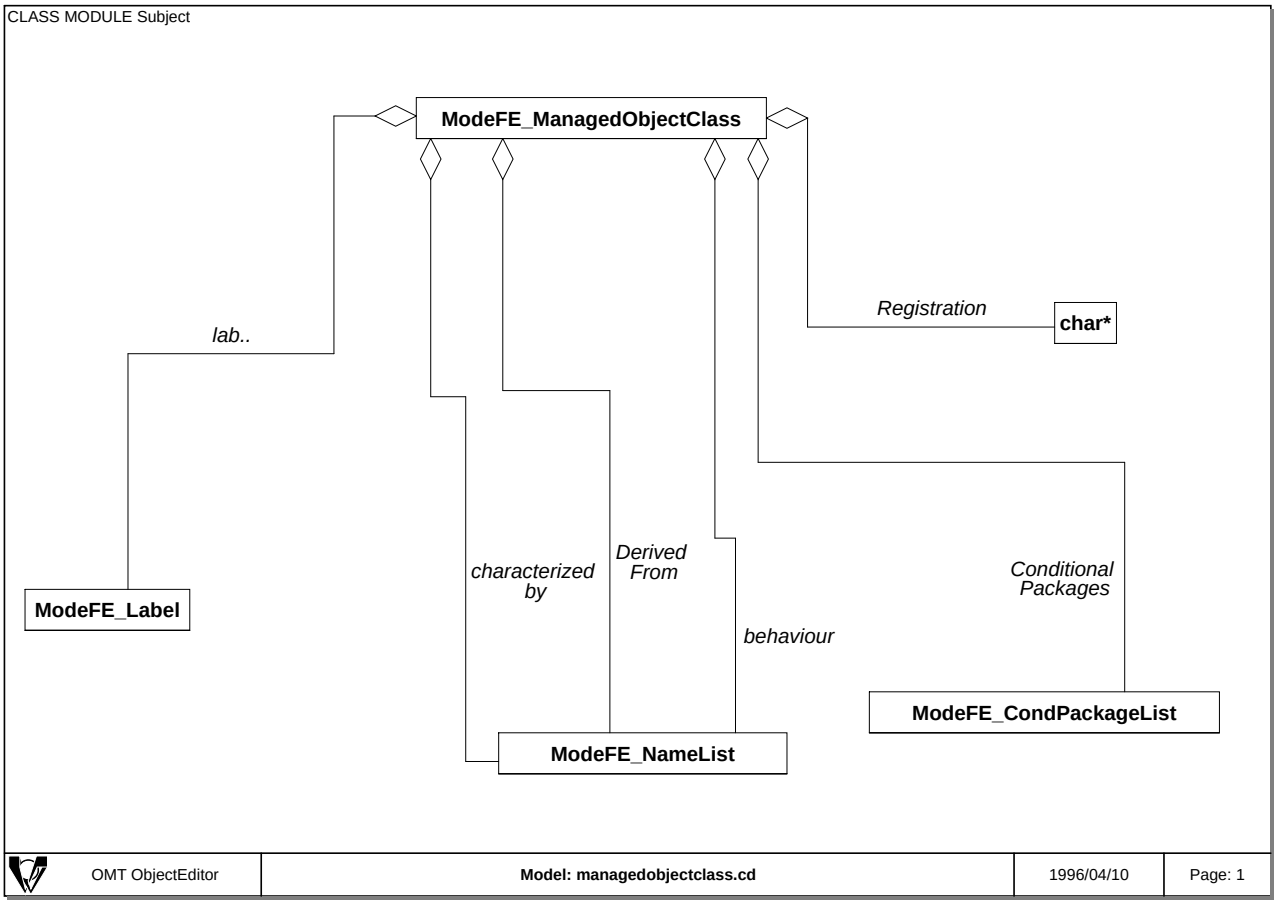
Prints in an ASCII format and a GDMO compliant form to the stream, the label and associated labels.

A.1.11 The ModeFE_ManagedObjectClass class

Purpose

The ModeFE_ManagedObjectClass class stores information related to the specification of a GDMO Managed Object Class (MO) type. It contains the MOs label, its registration identifier, the list of the super-classes from which the class is derived, the list of associated mandatory packages, the list of associated behaviour labels and the list of conditional packages

Architecture



Available methods

```
ModeFE_ManagedObjectClass (ModeFE_Label *,
                           char *,
                           ModeFE_NameList *,
                           ModeFE_NameList *,
                           ModeFE_NameList *,
                           ModeFE_CondPackageList *);
```

Basic constructor for an MO object. The parameters are:

1. the label associated with the MO specification,
2. the registration identifier of the definition without the "{" and "}" brackets,
3. the list of labels describing all super-classes from which this MO is derived,
4. the list of package labels which refer to all mandatory packages associated with the managed object class,
5. the list of behaviour labels which refer to behaviour descriptions associated with the MO,
6. the list of conditional packages associated with the MO.

```
ModeFE_ManagedObjectClass ();
```

Default constructor for the ModeFE_ManagedObjectClass class which creates an empty MO definition.

```
ModeFE_ManagedObjectClass (const ModeFE_ManagedObjectClass&);
```

Copy constructor of the ModeFE_ManagedObjectClass class. Returns no element in the current version of the library.

```
~ModeFE_ManagedObjectClass ();
```

Destructor of the ModeFE_ManagedObjectClass class. Deletes all contained elements.

```
ModeFE_Label* GetLabel();
```

Returns a pointer to a copy of the MO type label. If no label is associated, returns a pointer to an empty ModeFE_Label object.

```
void SetLabel(ModeFE_Label* );
```

Assigns a new value to the MO type label.

```
char* GetRegistration();
```

Returns a pointer to a copy of the MO type registration identifier. The identifier is provided as a string and does not contain the "{" and "}" brackets.

```
void SetRegistration(char*);
```

Assigns a new registration identifier to the MO specification. The registration identifier should not include the start and stop brackets ("{","}").

```
ModeFE_NameList* GetParentClasses();
```

Returns a pointer to a copy of the list of the labels of the super classes from which this MO is derived from. If no super-class exists (this shoul never be the case in the OSI framework), the method returns a empty list.

int AddParentClass(ModeFE_Label *);

Adds one label to the head of the list of super-classes associated with this MO. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AppendParentClass(ModeFE_Label *);

Adds one label to the end of the list of super-classes associated with this MO. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int RemoveParentClass(ModeFE_Label *);

Removes one label from the list of super-classes associated with this MO. This method returns `OK` if the operation completed successfully. It returns `ERROR` otherwise.

int ReplaceParentClass(ModeFE_Label *, ModeFE_Label*);

Replaces the first occurence of the first parameter in the list of super classes labels associated with the MO type with the label given as a second parameter. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

void ClearParentList();

Removes all super-classes labels from the list of parent classes labels associated with the MO type.

ModeFE_NameList* GetPackages();

Returns the labels of all package definitions associated with the MO type and described as mandatory in the MO. If no package is associated, then this methods returns an empty list.

int AddPackage(ModeFE_Label *);

Adds one package label to the head of the list of mandatory packages associated with the MO type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AppendPackage(ModeFE_Label *);

Adds one package label to the end of the list of mandatory packages associated with the MO type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int RemovePackage(ModeFE_Label *);

Removes the first occurence of a package label from the list of mandatory package labels associated with the MO type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplacePackage(ModeFE_Label *,ModeFE_Label*);

Replaces the first occurence of the first parameter in the list of mandatory package labels associated with the MO type with the label given as a second parameter. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

void ClearPackageList();

Removes all labels from the list of mandatory package labels associated with the MO type.

```
ModeFE_NameList* GetBehaviours();
```

Returns the labels of all behaviour definitions associated with the MO type. If no behaviour is associated, then this methods returns an empty list.

```
int AddBehaviour(ModeFE_Label *);
```

Adds one behaviour label to the head of the list of behaviours associated with the MO type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int AppendBehaviour(ModeFE_Label *);
```

Adds one behaviour label to the end of the list of behaviours associated with the MO type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int RemoveBehaviour(ModeFE_Label *);
```

Removes the first occurence of a behaviour label in the list of behaviour labels associated with the MO type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int ReplaceBehaviour(ModeFE_Label *, ModeFE_Label*);
```

Replaces the first occurence of the first parameter in the list of behaviour labels associated with the MO type with the label given as a second parameter. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
void ClearBehaviours();
```

Removes all behaviour labels from the list of behaviour labels associated with the MO type.

```
ModeFE_CondPackageList* GetCondPackages();
```

Returns the list of conditional packages associated with the MO definition. If the MO does not contain any ondotional packages the result s an empty list. See the ModeFE_CondPackage for the information available in the definition.

```
int AddCondPackage(ModeFE_CondPackage *);
```

Adds one conditional package label and condition to the head of the list of conditional packages associated with the MO type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int AppendCondPackage(ModeFE_CondPackage *);
```

Adds one conditional package label and condition to the end of the list of conditional packages associated with the MO type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int RemoveCondPackage(ModeFE_CondPackage *);
```

Removes the first occurence of the conditional package label from the list of conditional packages associated with the MO type. The selection is made only on the package label and not on the condition. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int ReplaceCondPackage(ModeFE_CondPackage * e, ModeFE_CondPackage *);
```

Replaces the first occurence of the conditional package label given in the first parameter from the list of conditional packages associated with the MO type with the second parameter. The selection is made only on the package label and not on the condition. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

void ClearCondPackageList();

Removes all conditional packages from the list of conditional packages associated with the MO type.

ModeFE_ManagedObjectClass* Dupplicate();

Returns a pointer to a copy of the current object. The invoker is then responsible for deleting this pointer and its content.

void Print (ofstream *) ;

Prints in an ASCII format to the stream, the GDMO specification that corresponds to this MO type specification.

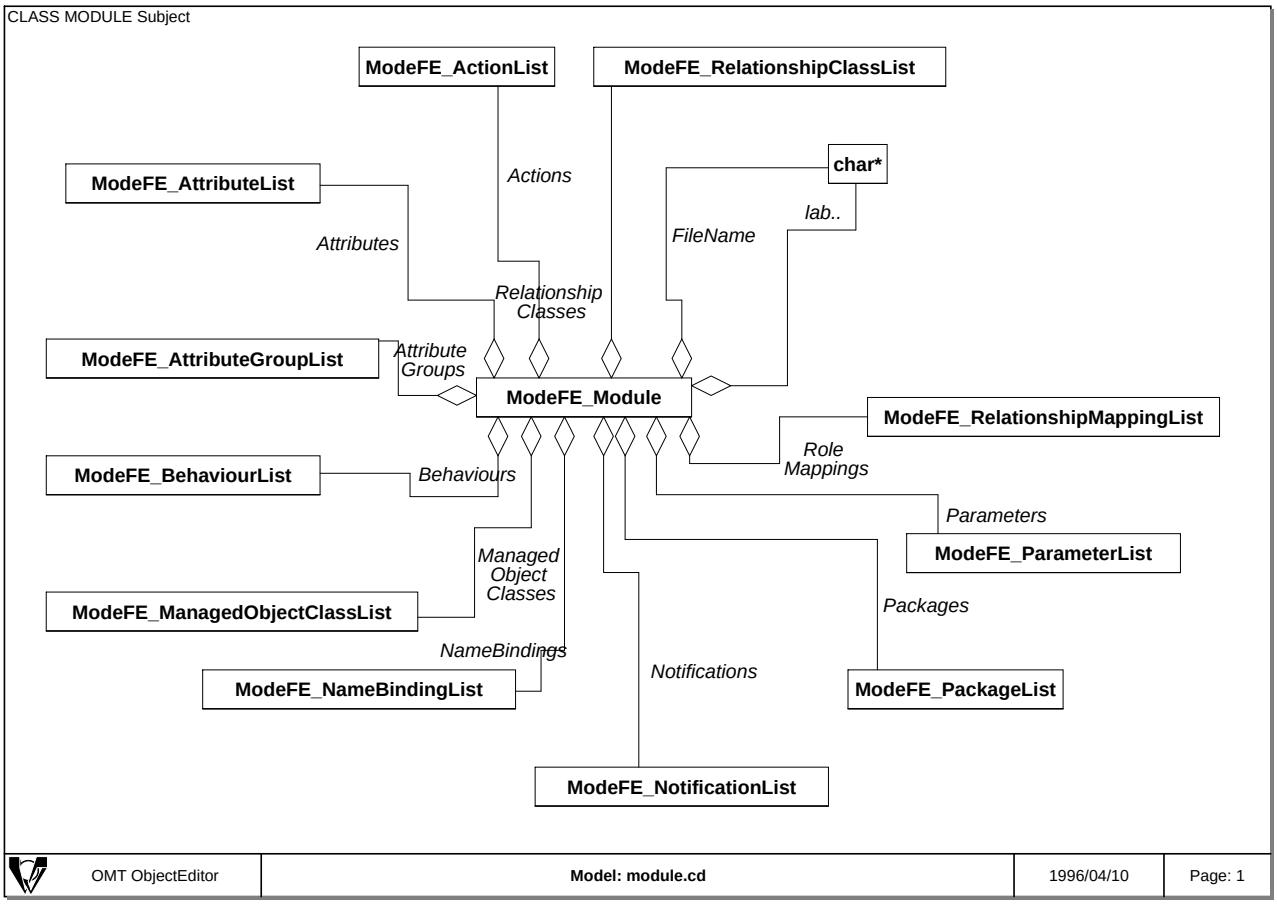
A.1.12 The ModeFE_Module class

Purpose

The ModeFE_Module Class stores information related to the specification of a set of GRM and GDMO specifiation. The module contains a module identifier, an associated specificcation file name, and several lists of specifications. These specifications are:

- managed object classes types,
- managed relationship types,
- packages types,
- action types,
- attribute types,
- attribute group types,
- behaviour definitions,
- name-binding types,
- notification types,
- parameter types,
- relationship mapping types.

Architecture



Available methods

```
ModeFE_Module( char* ,
               char* ,
               ModeFE_ActionList* ,
               ModeFE_AttributeList* ,
               ModeFE_AttributeGroupList* ,
               ModeFE_BehaviourList* ,
               ModeFE_ManagedObjectClassList* ,
               ModeFE_RelationshipClassList* ,
               ModeFE_NameBindingList* ,
               ModeFE_NotificationList* ,
               ModeFE_PackageList* ,
               ModeFE_ParameterList* ,
               ModeFE_RelationshipMappingList* );
```

Basic constructor of a ModeFE_Module object. The parameters are:

1. the module identifier (its label),
2. the file from which it was loaded,
3. the list of actions defined within the module,
4. the list of attributes defined within the module,
5. the list of attribute groups defined within the module,

6. the list of behaviours defined within the module,
7. the list of managed object classes defined within the module,
8. the list of relationship classes defined within the module,
9. the list of name-bindings defined within the module,
10. the list of notifications defined within the module,
11. the list of packages defined within the module,
12. the list of parameters defined within the module,
13. the list of relationship mappings defined within the module.

```
ModeFE_Module(char* , char* );
```

Cosntructor of the ModeFE_Module object. This constructor assigns the label and the filename and initializes to empty lists all contained specification lists.

```
ModeFE_Module();
```

Default constructor of the ModeFE_Module object. Initialises the label and filename to `NULL` and all lists to empty lists.

```
ModeFE_Module(const ModeFE_Module&);
```

Copy constructor of the ModeFE_Module class. Returns no element in the current version of the library.

```
~ModeFE_Module();
```

Destructor of the ModeFE_Module class. Deletes all contained elements.

```
char * GetLabel();
```

Returns a copy of the label associated with the module.

```
void SetLabel(char*);
```

Assigns a new module identifier to the module.

```
char * GetFileName();
```

Returns a copy of the filename from which this module was loaded.

```
void SetFileName(char*);
```

Assigns a new filename to the module.

```
ModeFE_ActionList* GetActions();
```

Returns the list of all action specifications defined within the module. If no action specification exists, the method returns an empty list.

```
ModeFE_AttributeList* GetAttributes();
```

Returns the list of all attribute specifications defined within the module. If no attribute specification exists, the method returns an empty list.

```
ModeFE_AttributeGroupList* GetAttributeGroups();
```

Returns the list of all attribute group specifications defined within the module. If no attribute group specification exists, the method returns an empty list.

```
ModeFE_BehaviourList* GetBehaviours();
```

Returns the list of all behaviour specifications defined within the module. If no behaviour specification exists, the method returns an empty list.

```
ModeFE_ManagedObjectClassList* GetManagedObjectClasses();
```

Returns the list of all managed object class specifications defined within the module. If no managed object class specification exists, the method returns an empty list.

```
ModeFE_RelationshipClassList* GetRelationshipClasses();
```

Returns the list of all relationship class specifications defined within the module. If no relationship class specification exists, the method returns an empty list.

```
ModeFE_NameBindingList* GetNameBindings();
```

Returns the list of all name-binding specifications defined within the module. If no name-binding specification exists, the method returns an empty list.

```
ModeFE_NotificationList* GetNotifications();
```

Returns the list of all notification specifications defined within the module. If no notification specification exists, the method returns an empty list.

```
ModeFE_PackageList* GetPackages();
```

Returns the list of all package specifications defined within the module. If no package specification exists, the method returns an empty list.

```
ModeFE_ParameterList* GetParameters();
```

Returns the list of all parameter specifications defined within the module. If no parameter specification exists, the method returns an empty list.

```
ModeFE_RelationshipMappingList* GetRelationshipMappings();
```

Returns the list of all relationship mapping specifications defined within the module. If no relationship mapping specification exists, the method returns an empty list.

```
void ClearActions();
```

Removes all action specifications from the module.

```
void ClearAttributes();
```

Removes all attribute specifications from the module.

void ClearAttributeGroups();

Removes all attribute group specifications from the module.

void ClearBehaviours();

Removes all behaviour specifications from the module.

void ClearManagedObjectClasses();

Removes all managed object class specifications from the module.

void ClearRelationshipClasses();

Removes all relationship class specifications from the module.

void ClearNameBindings();

Removes all name-binding specifications from the module.

void ClearNotifications();

Removes all notification specifications from the module.

void ClearPackages();

Removes all package specifications from the module.

void ClearParameters();

Removes all parameter specifications from the module.

void ClearRelationshipMappings();

Removes all relationship mapping specifications from the module.

int AddAction(ModeFE_Action*);

Adds one action specification to the head of the list of action specifications defined in this module. The method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AddAttribute(ModeFE_Attribute*);

Adds one attribute specification to the head of the list of attribute specifications defined in this module. The method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AddAttributeGroup(ModeFE_AttributeGroup*);

Adds one attribute group specification to the head of the list of attribute group specifications defined in this module. The method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AddBehaviour(ModeFE_Behaviour*);

Adds one behaviour specification to the head of the list of behaviour specifications defined in this module. The method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AddManagedObjectClass(ModeFE_ManagedObjectClass*);

Adds one managed object class specification to the head of the list of managed object class specifications defined in this module. The method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AddRelationshipClass(ModeFE_RelationshipClass*);

Adds one relationship class specification to the head of the list of relationship class specifications defined in this module. The method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AddNameBinding(ModeFE_NameBinding*);

Adds one name binding specification to the head of the list of name binding specifications defined in this module. The method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AddNotification(ModeFE_Notification*);

Adds one notification specification to the head of the list of notification specifications defined in this module. The method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AddPackage(ModeFE_Package*);

Adds one package specification to the head of the list of package specifications defined in this module. The method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AddParameter(ModeFE_Parameter*);

Adds one parameter specification to the head of the list of parameter specifications defined in this module. The method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AddRelationshipMapping(ModeFE_RelationshipMapping*);

Adds one relationship mapping specification to the head of the list of relationship mapping specifications defined in this module. The method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AppendAction(ModeFE_Action*);

Adds one action specification to the end of the list of action specifications defined in this module. The method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AppendAttribute(ModeFE_Attribute*);

Adds one attribute specification to the end of the list of attribute specifications defined in this module. The method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AppendAttributeGroup(ModeFE_AttributeGroup*);

Adds one attribute group specification to the end of the list of attribute group specifications defined in this module. The method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AppendBehaviour(ModeFE_Behaviour*);

Adds one behaviour specification to the end of the list of behaviour specifications defined in this module. The method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AppendManagedObjectClass(ModeFE_ManagedObjectClass*);

Adds one managed object class specification to the end of the list of managed object class specifications defined in this module. The method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AppendRelationshipClass(ModeFE_RelationshipClass*);

Adds one relationship class specification to the end of the list of relationship class specifications defined in this module. The method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AppendNameBinding(ModeFE_NameBinding*);

Adds one name-binding specification to the end of the list of name-binding specifications defined in this module. The method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AppendNotification(ModeFE_Notification*);

Adds one notification specification to the end of the list of notification specifications defined in this module. The method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AppendPackage(ModeFE_Package*);

Adds one package specification to the end of the list of package specifications defined in this module. The method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AppendParameter(ModeFE_Parameter*);

Adds one parameter specification to the end of the list of parameter specifications defined in this module. The method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AppendRelationshipMapping(ModeFE_RelationshipMapping*);

Adds one relationship mapping specification to the end of the list of relationship mapping specifications defined in this module. The method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int RemoveAction(ModeFE_Action*);

Removes the first occurence of an action specification which has the same label as the action specification given as a parameter to the method, from the list of action specifications defined in the module. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int RemoveAttribute(ModeFE_Attribute*);

Removes the first occurence of an attribute specification which has the same label as the attribute specification given as a parameter to the method, from the list of attribute specifications defined in the module. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int RemoveAttributeGroup(ModeFE_AttributeGroup*);

Removes the first occurence of an attribute group specification which has the same label as the attribute group specification given as a parameter to the method, from the list of attribute group specifications defined in the module. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int RemoveBehaviour(ModeFE_Behaviour*);

Removes the first occurence of a behaviour specification which has the same label as the behaviour specification given as a parameter to the method, from the list of behaviour specifications defined in the module. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int RemoveManagedObjectClass(ModeFE_ManagedObjectClass*);

Removes the first occurence of a managed object class specification which has the same label as the managed object class specification given as a parameter to the method, from the list of managed object class specifications defined in the module. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int RemoveRelationshipClass(ModeFE_RelationshipClass*);

Removes the first occurence of a relationship class specification which has the same label as the relationship class specification given as a parameter to the method, from the list of relationship class specifications defined in the module. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int RemoveNameBinding(ModeFE_NameBinding*);

Removes the first occurence of a name-binding specification which has the same label as the name-binding specification given as a parameter to the method, from the list of name-binding specifications defined in the module. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int RemoveNotification(ModeFE_Notification*);

Removes the first occurence of a notification specification which has the same label as the notification specification given as a parameter to the method, from the list of notification specifications defined in the module. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int RemovePackage(ModeFE_Package*);

Removes the first occurence of a package specification which has the same label as the package specification given as a parameter to the method, from the list of package specifications defined in the module. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int RemoveParameter(ModeFE_Parameter*);

Removes the first occurence of a parameter specification which has the same label as the parameter specification given as a parameter to the method, from the list of parameter specifications defined in the module. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int RemoveRelationshipMapping(ModeFE_RelationshipMapping*);

Removes the first occurence of a relationship mapping specification which has the same label as the relationship mapping specification given as a parameter to the method, from the list of relationship mapping specifications defined in the module. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplaceAction(ModeFE_Action*,ModeFE_Action*);

Replaces the first action specification occurence of the first parameter given for the action method in the list of action specifications defined in the module with the action specification given as the second parameter to the method. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplaceAttribute(ModeFE_Attribute*,ModeFE_Attribute*);

Replaces the first attribute specification occurence of the first parameter given for the method in the list of attribute specifications defined in the module with the attribute specification given as the second parameter to the method. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplaceAttributeGroup(ModeFE_AttributeGroup*,ModeFE_AttributeGroup*);

Replaces the first attribute group specification occurence of the first parameter given for the method in the list of attribute group specifications defined in the module with the attribute group specification given as the second parameter to the method. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplaceBehaviour(ModeFE_Behaviour*,ModeFE_Behaviour*);

Replaces the first behaviour specification occurence of the first parameter given for the method in the list of behaviour specifications defined in the module with the behaviour specification given as the second parameter to the method. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplaceManagedObjectClass(ModeFE_ManagedObjectClass*,ModeFE_ManagedObjectClass*);

Replaces the first managed object class specification occurence of the first parameter given for the method in the list of managed object class specifications defined in the module with the managed object class specification given as the second parameter to the method. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplaceRelationshipClass(ModeFE_RelationshipClass*,ModeFE_RelationshipClass*);

Replaces the first relationship class specification occurence of the first parameter given for the method in the list of relationship class specifications defined in the module with the relationship class specification given as the second parameter to the method. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplaceNameBinding(ModeFE_NameBinding*, ModeFE_NameBinding*);

Replaces the first name-binding specification occurence of the first parameter given for the method in the list of name-binding specifications defined in the module with the name-binding specification given as the second parameter to the method. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplaceNotification(ModeFE_Notification*,ModeFE_Notification*);

Replaces the first notification specification occurence of the first parameter given for the method in the list of notification specifications defined in the module with the notification specification given as the second parameter to the method. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplacePackage(ModeFE_Package*,ModeFE_Package*);

Replaces the first package specification occurence of the first parameter given for the method in the list of package specifications defined in the module with the package specification given as the second parameter to the method. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplaceParameter(ModeFE_Parameter*,ModeFE_Parameter*);

Replaces the first parameter specification occurence of the first parameter given for the method in the list of parameter specifications defined in the module with the parameter specification given as the second parameter to the method. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplaceRelationshipMapping(ModeFE_RelationshipMapping*,ModeFE_RelationshipMapping*);

Replaces the first relationship mapping specification occurence of the first parameter given for the method in the list of relationship mapping specifications defined in the module with the relationship mapping specification given as the second parameter to the method. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int IsEqual(ModeFE_Module*);

Returns `TRUE` if the module given as a parameter to the method is equal to the current module. Returns `FALSE` otherwise. The equlity test is made only on the module identifier in the current version of the library.

ModeFE_Module* Dupplicate();

Returns a pointer to a copy of the current module.

void Print (ofstream *) ;

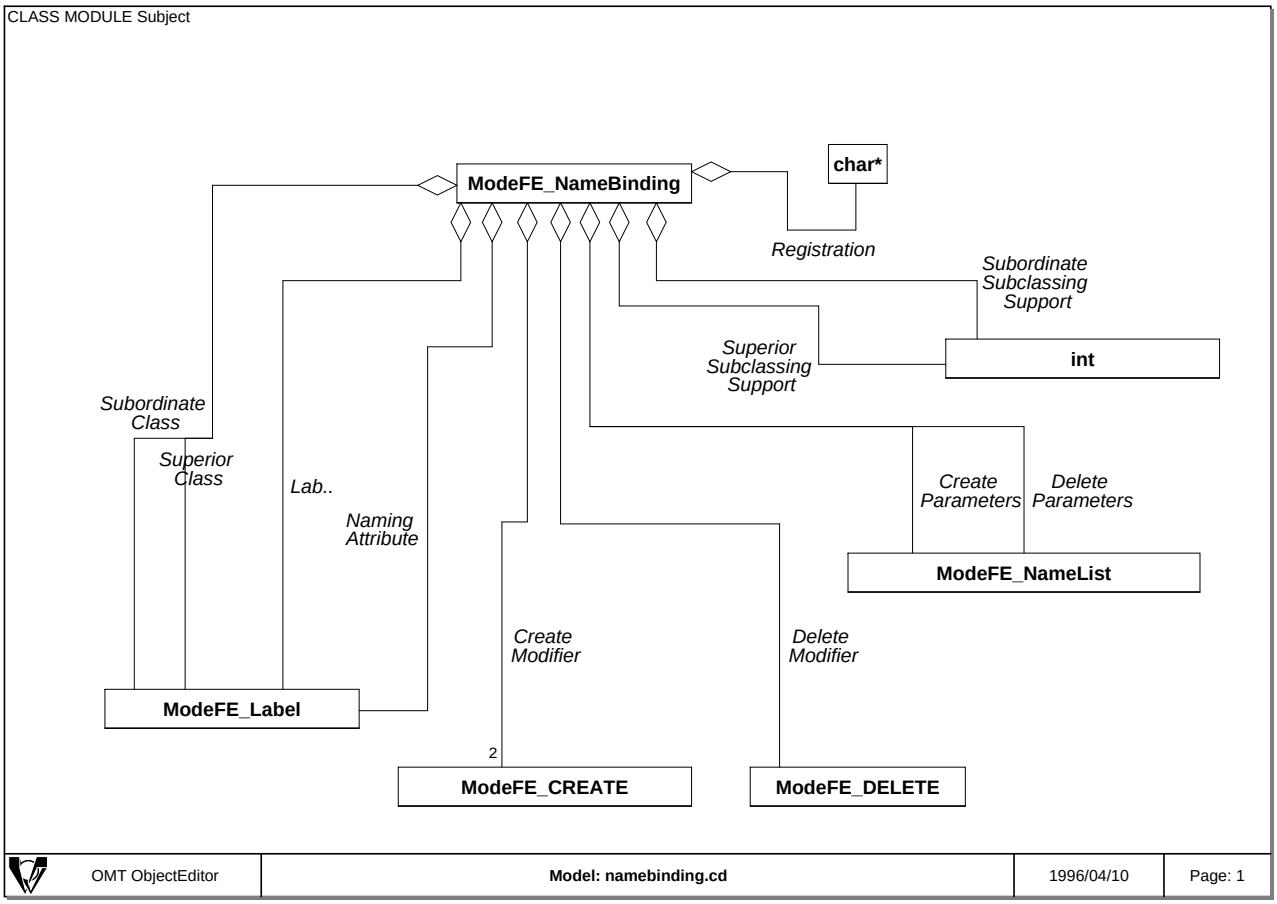
Prints in an ASCII format to the stream, the GDMO and GRM specifications that are contained within the module.

A.1.13 The ModeFE_NameBinding class

Purpose

The ModeFE_NameBinding Class stores information related to the specification of a GDMO Name-Binding type. It contains the Name-Binding label, its registration identifier, the list of associated behaviour labels, the label of the superior object class, the label of the subordinate object class, statements over subordinate and/or superior subclassing support, the label of the attribute of the subordinate class used for naming and creation and deletion information.

Architecture



Available methods

```
ModeFE_NameBinding ( ModeFE_Label * ,
                     char * ,
                     ModeFE_Label * ,
                     int ,
                     ModeFE_Label * ,
                     int ,
                     ModeFE_Label * ,
                     ModeFE_NameList * ,
                     ModeFE_CREATE ,
                     ModeFE_CREATE ,
                     ModeFE_NameList * ,
                     ModeFE_DELETE ,
                     ModeFE_NameList * );
```

Basic constructor for the ModeFE_NameBinding class. The parameters of this method are:

1. the Name-Bindings label,
2. its registration identifier
3. the label of the subordinate class,
4. a statement on subordinate subclassing support by the Name-Binding:

- 0: no suclassing allowed,
- 1: subclassing allowed.

5. the label of the superior class,
6. a statement on superior subclassing support by the Name-Binding:
   - 0: no suclassing allowed,
   - 1: subclassing allowed.
7. the label of the attribute used for naming,
8. the list of behaviour labels associated with the Name-Binding,
9. the first create modifier. Possible values are:
   - `NOCREATE_`: no modifier is defined,
   - `WITH_REFERENCE_OBJECT_`,
   - `WITH_AUTOMATIC_INSTANCE_NAMING_`
10. the second create modifier. Possible values are:
    - `NOCREATE_`: no modifier is defined,
    - `WITH_REFERENCE_OBJECT_`,
    - `WITH_AUTOMATIC_INSTANCE_NAMING_`

    If only one create modifier exists for the Name-Binding, then the first one must carry the information and the second one must be equal to `NOCREATE_`
11. the list of parameters, if any, associated with the create operation,
12. the delete modifier of the Name-Binding. Possible values are:
    - `NODELETE_` (no delete modifier is specified),
    - `ONLY_IF_NO_CONTAINED_OBJECTS_`,
    - `DELETES_CONTAINED_OBJECTS_`
13. the list of parameters, if any, associated with the delete operation.

```
ModeFE_NameBinding ();
```

Default constructor for the ModeFE_NameBinding class which creates an empty ModeFE_NameBinding definition.

```
ModeFE_NameBinding (const ModeFE_NameBinding&);
```

Copy constructor of the ModeFE_NameBinding class. Returns no element in the current version of the library.

```
~ModeFE_NameBinding ();
```

Destructor of the ModeFE_NameBinding class. Deletes all contained elements.

```
ModeFE_Label* GetLabel();
```

Returns a pointer to a copy of the Name-Binding label. If no label is associated, returns a pointer to an empty ModeFE_Label object.

void SetLabel(ModeFE_Label*);

Assigns a new value to the label of the Name-Binding.

char* GetRegistration();

Returns a pointer to a copy of the registration identifier of the Name-Binding. The identifier is provided as a string and does not contain the "{" and "}" brackets.

void SetRegistration(**char***);

Assigns a new registration identifier to the Name-Binding specification. The registration identifier should not include the start and stop brackets ("{","}").

ModeFE_Label* GetSubordinateClassName();

Returns the label of the managed object class which plays the subordinate role in the Name-Binding.

void SetSubordinateClassName(ModeFE_Label *);

Assigns a new value to the label which identifies the class playing the subordinate role in this Name-Binding.

int GetSubordinateSubclassesSupport();

Returns 1 if subclasses of the subodinate class are allowed to be subordinate classes in the context of this Name-Binding. Returns 0 if not.

void SetSubordinateSubclassesSupport(**int**);

Assigns the statement of subclassing support for the subordinate role in the Name-Binding:

- 1: subclassing is allowed,
- 0: subclassing is forbidden.

ModeFE_Label* GetSuperiorClass();

Returns the label of the managed object class which plays the superior role in the Name-Binding. **void** SetSuperiorClass(Mod

Assigns a new value to the label which identifies the class playing the superior role in this Name-Binding.

int GetSuperiorSubclassesSupport();

Returns 1 if subclasses of the superior class are allowed to be superior classes in the context of this Name-Binding. Returns 0 if not.

void SetSuperiorSubclassesSupport(**int**);

Assigns the statement of subclassing support for the superior role in the Name-Binding:

- 1: subclassing is allowed,
- 0: subclassing is forbidden.

ModeFE_Label* GetAttribute();

Returns the label of the attribute used for naming purpose in the Name-Binding.

void SetAttribute(ModeFE_Label*);

Assigns a new value to the label describing the attribute used for naming in the context of this Name-Binding.

ModeFE_NameList* GetBehaviours();

Returns the labels of all behaviour definitions associated with the Name-Binding. If no behaviour is associated, then this methods returns an empty list.

int AddBehaviour(ModeFE_Label *);

Adds one behaviour label to the head of the list of behaviours associated with the Name-Binding. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AppendBehaviour(ModeFE_Label *);

Adds one behaviour label to the end of the list of behaviours associated with the Name-Binding. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int RemoveBehaviour(ModeFE_Label *);

Removes the first occurence of a behaviour label in the list of behaviour labels associated with the Name-Binding. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplaceBehaviour(ModeFE_Label *, ModeFE_Label *);

Replaces the first occurence of the first parameter in the list of behaviour labels associated with the Name-Binding with the label given as a second parameter. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

void ClearBehaviours();

Removes all labels from the list of behaviour labels associated with this Name-Binding definition.

ModeFE_CREATE GetCreateModifier1();

Returns the first create modifier associated with the Name-Binding. Possible values are:

- `NOCREATE_`: no modifier is defined,
- `WITH_REFERENCE_OBJECT_`,
- `WITH_AUTOMATIC_INSTANCE_NAMING_`

void SetCreateModifier1(ModeFE_CREATE);

Assigns a new value to the first create modifier of the Name-Binding. Possible values for the parameter are:

- `NOCREATE_`: no modifier is defined,
- `WITH_REFERENCE_OBJECT_`,
- `WITH_AUTOMATIC_INSTANCE_NAMING_`

ModeFE_CREATE GetCreateModifier2();

Returns the second create modifier associated with the Name-Binding. Possible values are:

- `NOCREATE_`: no modifier is defined,
- `WITH_REFERENCE_OBJECT_`,
- `WITH_AUTOMATIC_INSTANCE_NAMING_`

void SetCreateModifier2(ModeFE_CREATE);

Assigns a new value to the second create modifier of the Name-Binding. Possible values for the parameter are:

- `NOCREATE_`: no modifier is defined,
- `WITH_REFERENCE_OBJECT_`,
- `WITH_AUTOMATIC_INSTANCE_NAMING_`

ModeFE_DELETE GetDeleteModifier();

Returns the value of the delete modifier which holds for the Name-Binding. Possible values are:

- `NODELETE_` (no delete modifier is specified),
- `ONLY_IF_NO_CONTAINED_OBJECTS_`,
- `DELETES_CONTAINED_OBJECTS_`

void SetDeleteModifier(ModeFE_DELETE);

Assigns a new value to the delete modifier of the Name-Binding. Possible values for the parameter are:

- `NODELETE_` (no delete modifier is specified),
- `ONLY_IF_NO_CONTAINED_OBJECTS_`,
- `DELETES_CONTAINED_OBJECTS_`

ModeFE_NameList* GetCreateParameters();

Returns the labels of all parameter definitions associated with the create operation of the Name-Binding. If no parameter is associated, then this methods returns an empty list.

int AddCreateParameter(ModeFE_Label *);

Adds one parameter label to the head of the list of parameters associated with the create operation of the Name-Binding. The method returns `OK` if the operation was successful, `ERROR` otherwise.

int AppendCreateParameter(ModeFE_Label *);

Adds one parameter label to the end of the list of parameters associated with the create operation of the Name-Binding. The method returns `OK` if the operation was successful, `ERROR` otherwise.

int RemoveCreateParameter(ModeFE_Label *);

Removes the first occurence of a parameter label in the list of parameter labels associated with the create operation of the Name-Binding. The method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplaceCreateParameter(ModeFE_Label *, ModeFE_Label*);

Replaces the first occurence of the first parameter in the list of parameter labels associated with the create operation of the Name-Binding type with the label given as a second parameter. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

void ClearCreateParameters();

Removes all parameters associated with the create operation of the Name-Binding.

ModeFE_NameList* GetDeleteParameters();

Returns the labels of all parameter definitions associated with the delete operation of the Name-Binding. If no parameter is associated, then this methods returns an empty list.

int AddDeleteParameter(ModeFE_Label *);

Adds one parameter label to the head of the list of parameters associated with the delete operation of the Name-Binding. The method returns `OK` if the operation was successful, `ERROR` otherwise.

int AppendDeleteParameter(ModeFE_Label *);

Adds one parameter label to the end of the list of parameters associated with the delete operation of the Name-Binding. The method returns `OK` if the operation was successful, `ERROR` otherwise.

int RemoveDeleteParameter(ModeFE_Label *);

Removes the first occurence of a parameter label in the list of parameter labels associated with the delete operation of the Name-Binding. The method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplaceDeleteParameter(ModeFE_Label *, ModeFE_Label*);

Replaces the first occurence of the first parameter in the list of parameter labels associated with the delete operation of the Name-Binding type with the label given as a second parameter. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

void ClearDeleteParameters();

Removes all parameters associated with the create operation of the Name-Binding.

int IsEqual(ModeFE_NameBinding*);

Returns `TRUE` if the Name-Binding given in parameter to the method is has the same label as the current Name-Binding.

ModeFE_NameBinding* Dupplicate();

Returns a pointer to a copy of the current object. The invoker is then responsible for deleting this pointer and its content.

```
void Print( ofstream * ) ;
```

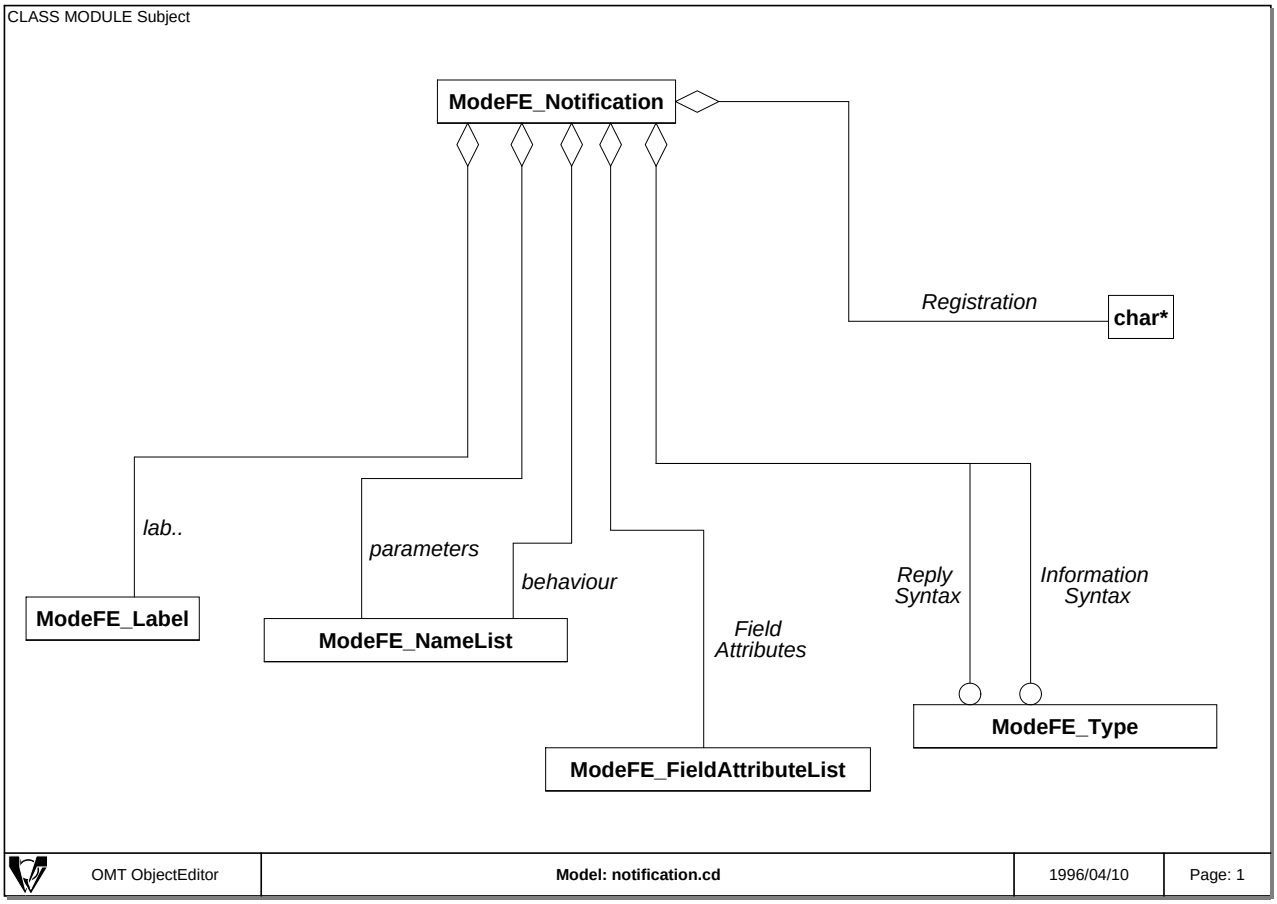
Prints in an ASCII format to the stream, the GDMO specification that corresponds tot this Name-Binding specification.

A.1.14 The ModeFE_Notification class

Purpose

The ModeFE_Notification Class stores information related to the specification of a GDMO Notification type. It contains the notification label, its registration identifier, the list of associated behaviour labels, the list of associated parameters and the corresponding fields, the information syntax if specified, the link to attributes and its reply syntax if specified and the invocation mode (confirmed or not).

Architecture



Available methods

```
ModeFE_Notification ( ModeFE_Label *,
                      char * ,
                      ModeFE_NameList * ,
                      ModeFE_NameList * ,
                      ModeFE_Type ,
                      ModeFE_FieldAttributeList * ,
                      ModeFE_Type );
```

Basic constructor for a notification object. The parameters are:

1. the label associated with the notification specification,

2. the registration identifier of the definition without the "{" and "}" brackets,
3. the list of associated behaviour labels,
4. the list of associated parameter labels,
5. the information syntax (`NULL` if none),
6. the list of associated attribute and the respective fields in the information syntax,
7. the reply syntax (`NULL` if none).

```
ModeFE_Notification ();
```

Default constructor for the ModeFE_Notification class which creates an empty ModeFE_Notification definition.

```
ModeFE_Notification (const ModeFE_Notification&);
```

Copy constructor of the ModeFE_Notification class. Returns no element in the current version of the library.

```
~ModeFE_Notification ();
```

Destructor of the ModeFE_Notification class. Deletes all contained elements.

```
ModeFE_Label* GetLabel();
```

Returns a pointer to a copy of the notification type label. If no label is associated, returns a pointer to an empty ModeFE_Label object.

```
int SetLabel(ModeFE_Label*);
```

Assigns a new value to the notification type label. Returns `OK` if the operation was successful. Returns `ERROR` otherwise.

```
char* GetRegistration();
```

Returns a pointer to a copy of the notification type registration identifier. The identifier is provided as a string and does not contain the "{" and "}" brackets.

```
void SetRegistration(char*);
```

Assigns a new registration identifier to the notification specification. The registration identifier should not include the start and stop brackets ("{","}").

```
ModeFE_NameList* GetBehaviours();
```

Returns the labels of all behaviour definitions associated with the notification type. If no behaviour is associated, then this methods returns an empty list.

```
int AddBehaviour(ModeFE_Label *);
```

Adds one behaviour label to the head of the list of behaviours associated with the notification type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int AppendBehaviour(ModeFE_Label *);
```

Adds one behaviour label to the end of the list of behaviours associated with the notification type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int RemoveBehaviour(ModeFE_Label *);
```

Removes the first occurence of a behaviour label in the list of behaviour labels associated with the notification type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int ReplaceBehaviour(ModeFE_Label *, ModeFE_Label*);
```

Replaces the first occurence of the first parameter in the list of behaviour labels associated with the notification type with the label given as a second parameter. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
void ClearBehaviours();
```

Removes all labels from the list of behaviour labels associated with this notification definition.

```
ModeFE_Type GetInformationSyntax();
```

Returns the information syntax of the notification type. If no information syntax was specified for the notification type, the method returns `NULL`.

```
int SetInformationSyntax(ModeFE_Type);
```

Assigns a new information syntax to the notification type. This value can be either a character string or `NULL` if no syntax is defined. The method returns `OK` if the operation was successful, `FALSE` otherwise.

```
ModeFE_Type GetReplySyntax();
```

Returns the reply syntax of the notification type . If no reply syntax was specified for the notification type, the method returns `NULL`.

```
int SetReplySyntax(ModeFE_Type);
```

Assigns a new reply syntax to the notification type. This value can be either a character string or `NULL` if no syntax is defined. The method returns `OK` if the operation was successful, `FALSE` otherwise.

```
ModeFE_NameList* GetParameterList();
```

Returns the list of parameter labels associated with the notification type. If no parameter is specified then the methods returns an empty list (see section on lists for details).

```
int AddParameter(ModeFE_Label*);
```

Adds one parameter label to the head of the list of parameters associated with the notification type. The methods returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int AppendParameter(ModeFE_Label*);
```

Adds one parameter label to the end of the list of parameters associated with the notification type. The methods returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int RemoveParameter(ModeFE_Label *);
```

Removes the first occurence of the parameter label given in parameter in the list of parameter labels associated with the notification type. The methods returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplaceParameter(ModeFE_Label*, ModeFE_Label *);

Replaces the first occurence of the parameter label given in the first parameter with the second parameter in the list of parameter labels associated with the notification type. The methods returns `OK` if the operation was successful. It returns `ERROR` otherwise.

void ClearParameters();

Removes all parameter labels from the list of parameters associated with the notification type.

ModeFE_FieldAttributeList* GetAttributeList();

Returns the list of attributes and their link to the fields of information syntax associated with the notification. If no attribute and field association is specified, the method returns an empy list.

int AddAttribute(ModeFE_FieldAttribute*);

Adds one attribute label and the associated field (the field that carries the value of the attribute) to the head of the list of attribute identifiers associated with the notification. The methods returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AppendAttribute(ModeFE_FieldAttribute*);

Adds one attribute label and the associated field (the field that carries the value of the attribute) to the end of the list of attribute identifiers associated with the notification. The methods returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int RemoveAttribute(ModeFE_FieldAttribute *);

Removes the first occurence of an attribute and associated field labels from the list of attribute and fields associated with the notification type. The selection is made only on the attributes label and not on the field name basis. The methods returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplaceAttribute(ModeFE_FieldAttribute*, ModeFE_FieldAttribute *);

Replace the first occurence of an attribute and associated field labels that corresponds to the first parameter of the method with the association given as the second parameter in the list of attribute and fields associated with the notification type. The selection is made only on the attributes label and not on the field name basis. The methods returns `OK` if the operation was successful. It returns `ERROR` otherwise.

void ClearAttributes();

Removes all attribute and associated field labels from the list of attribute and fields associated with the notificaction type.

ModeFE_Notification* Dupplicate();

Returns a pointer to a copy of the current object. The invoker is then responsible for deleting this pointer and its content.

int IsEqual(ModeFE_Notification*);

Returns `TRUE` if the notification given in parameter is equal to the current one. Returns `FALSE` otherwise. The equality test is made only on the notification label in this version of the library.

void Print(ofstream *) ;

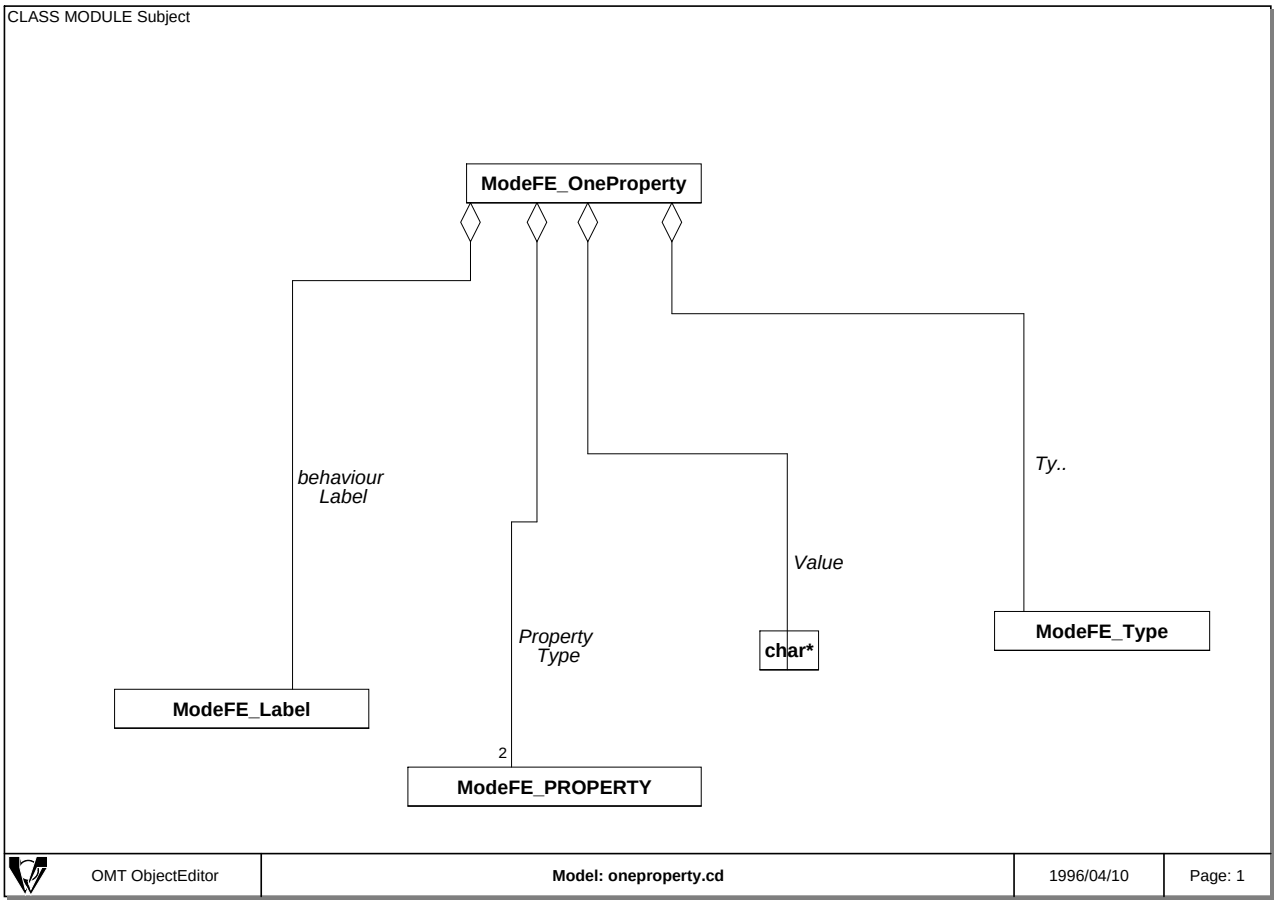
Prints in an ASCII format to the stream, the GDMO specification that corresponds to this notification type specification.

A.1.15 The ModeFE_OneProperty class

Purpose

The ModeFE_OneProperty Class stores information related to the specification of properties associated to an attribute specification in a package description. This information is the property which can be a keyword or a keyword associated with either a type spcifier, a value specifier or a behaviour description.

Architecture



Available methods

ModeFE_OneProperty (ModeFE_PROPERTY , **char** * , ModeFE_Type, ModeFE_Label*);

Basic constructor of the ModeFE_OneProperty class. The parameters are:

1. the property type. Possible values are:

- REPLACE_WITH_DEFAULT_
- DEFAULT_VALUE_
- INITIAL_VALUE_
- PERMITTED_VALUES_
- REQUIRED_VALUES_
- REQUIRED_VALUES_DERIVATION_
- DEFAULT_VALUE_DERIVATION_
- PERMITTED_VALUES_DERIVATION_
- INITIAL_VALUE_DERIVATION_
- GET_
- REPLACE_
- GET_REPLACE_
- ADD_
- REMOVE_
- ADD_REMOVE_
- NoProperty_

2. the value descriptor if any,
3. the reference type, if any,
4. the associated behaviour label, if any.

ModeFE_OneProperty ();

Default constructor of the ModeFE_OneProperty. Creates an empty property object.

ModeFE_OneProperty (**const** ModeFE_OneProperty&);

Copy constructor of the ModeFE_OneProperty class. Returns no element in the current version of the library.

~ModeFE_OneProperty ();

Destructor of the ModeFE_OneProperty class. Deletes all contained elements.

ModeFE_PROPERTY GetProperty();

Returns the type of the property. Possible results are:

- REPLACE_WITH_DEFAULT_
- DEFAULT_VALUE_
- INITIAL_VALUE_
- PERMITTED_VALUES_
- REQUIRED_VALUES_
- REQUIRED_VALUES_DERIVATION_
- DEFAULT_VALUE_DERIVATION_
- PERMITTED_VALUES_DERIVATION_

- INITIAL_VALUE_DERIVATION_
- GET_
- REPLACE_
- GET_REPLACE_
- ADD_
- REMOVE_
- ADD_REMOVE_
- NoProperty_

void SetProperty(ModeFE_PROPERTY);

Assigns a new value to the property type. Possible values for the parameter are:

- REPLACE_WITH_DEFAULT_
- DEFAULT_VALUE_
- INITIAL_VALUE_
- PERMITTED_VALUES_
- REQUIRED_VALUES_
- REQUIRED_VALUES_DERIVATION_
- DEFAULT_VALUE_DERIVATION_
- PERMITTED_VALUES_DERIVATION_
- INITIAL_VALUE_DERIVATION_
- GET_
- REPLACE_
- GET_REPLACE_
- ADD_
- REMOVE_
- ADD_REMOVE_
- NoProperty_

char* GetValueSpecifier();

Returns a pointer to a copy of the name of the value specifier associated with the property. If no value specifier exists, the method returns NULL.

int SetValueSpecifier(**char***);

Assigns a new name to the value specifier associated with the property. If one wants no value specifier to tbe associated with the property, the parameter value shall be NULL.

ModeFE_Type GetTypeSpecifier();

Returns a pointer to a copy of the type specifier associated with the property. If no type specifier exists for this property, the method returns `NULL`.

```
int SetTypeSpecifier(ModeFE_Type);
```

Assigns a new type to the type specifier associated with the property. If one wants no type specifier to tbe associated with the property, the parameter value shall be `NULL`.

```
void SetBehaviourLabel(ModeFE_Label*);
```

Assigns a new behaviour label to the property if required. If no behaviour label has to associated with the property the parameter must be either `NULL` or an empty label.

```
ModeFE_Label* GetBehaviourLabel();
```

Returns a pointer to a copy of the behaviour label associated with the property. If no behaviour label exists, the method returns an empty label.

```
ModeFE_OneProperty* Dupplicate();
```

Returns a pointer to a copy of the current object. The invoker is then responsible for deleting this pointer and its content.

```
void Print ( ofstream * ) ;
```

Prints in an ASCII format to the stream, the GDMO specification that corresponds tot this property type specification.

A.1.16 The ModeFE_OperationMapping class

Purpose

The ModeFE_OperationMapping class stores information related to the specification of operation mappings within a relationship mapping specification. Information over one operation mapping covers the relationship operation features and the associated systems operations with their links to roles of the relationship.

Available methods

```
ModeFE_OperationMapping (ModeFE_RelationshipOperation * ,
                         ModeFE_SmoRoleList * );
```

Basic constructor for the ModeFE_OperationMapping class. The parameters of the constructor are:

1. one relationship operation specification,
2. the list of associated system management operations and their link to roles of the relationship.

```
ModeFE_OperationMapping();
```

Default constructor of the ModeFE_OperationMapping class. Creates an empty operation mapping specification.

```
ModeFE_OperationMapping(const ModeFE_OperationMapping&);
```

Copy constructor of the ModeFE_OperationMapping class. Returns no element in the current version of the library.

~ModeFE_OperationMapping();

Destructor of the ModeFE_OperationMapping class. Deletes all contained elements.

ModeFE_RelationshipOperation* GetRelationshipOperation();

Returns a pointer to a copy of the relationship operation that corresponds to this operation mapping.

int SetRelationshipOperation(ModeFE_RelationshipOperation*);

Assigns a new relationship operation to this operaion mapping. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

ModeFE_SmoRoleList* GetMappings();

Returns the list of system operations and their link to roles of the relationship that correspond to the mapping of this relationship operation. If no mapping exists, the method returns an empty list.

int SetMappings(ModeFE_SmoRoleList*);

Assigns a list of system operations and their link to roles of the relationship to the list of system operations that correspond to the mapping of the relationship operation. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AddMapping(ModeFE_SmoRole *);

Adds one system operations and its link to roles of the relationship to the head of the list of system operations that correspond to the mapping of the relationship operation. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AppendMapping(ModeFE_SmoRole *);

Adds one system operations and its link to roles of the relationship to the end of the list of system operations that correspond to the mapping of the relationship operation. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int RemoveMapping(ModeFE_SmoRole *);

Removes the first occurence of the system operation and its link to roles of the relationship that has the same label as the parameter of the method with the parameter in the list of system operations that correspond to the mapping of the relationship operation. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplaceMapping(ModeFE_SmoRole *, ModeFE_SmoRole*);

Replaces the first occurence of the first parameter in the list of system operation mappings associated with the relationship operation mapping with the system operation given as a second parameter. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

void ClearMappings();

Removes all system operation mappings that correspond to this relationship operation in the list of mappings.

int IsEqual(ModeFE_OperationMapping*);

Returns `TRUE` if the operatoion mapping given in parameter is equal to the current one. Returns `FALSE` otherwise. The equality test is made only on the relationship operation name in this version of the library.

ModeFE_OperationMapping* Dupplicate();

Returns a pointer to a copy of the current object. The invoker is then responsible for deleting this pointer and its content.

void Print(ofstream *);

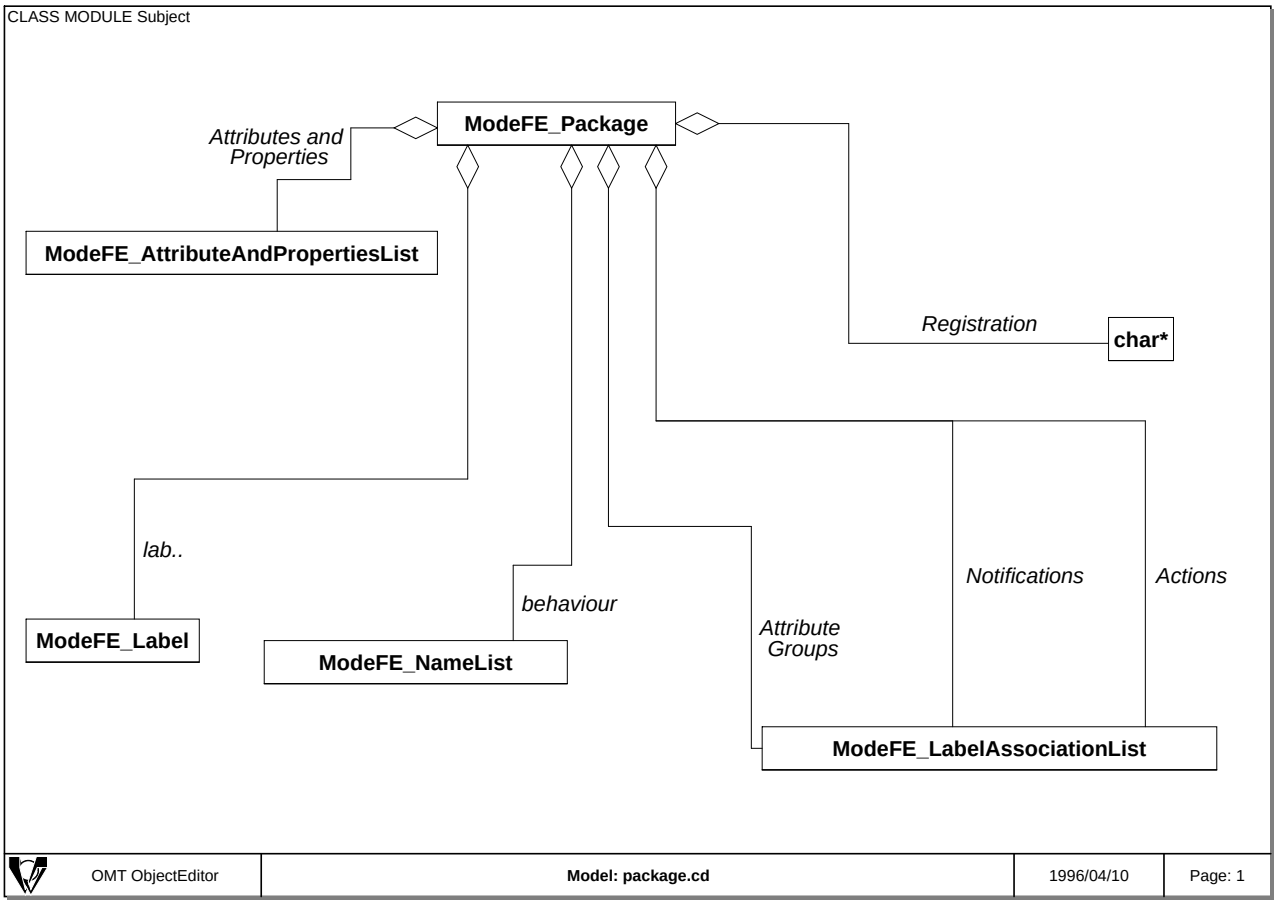
Prints in an ASCII format to the stream, the GDMO specification that corresponds to this operation mapping type specification.

A.1.17 The ModeFE_Package class

Purpose

The ModeFE_Package class stores information related to a GDMO package specification. An instance contains the package label, its registration identifier, the labels of the behaviour definitions related to the package, the list of attributes and the associated properties in the package, the attribute groups and their elements in the package, the actions and notifications defined in the package together with their parameters.

Architecture



Available methods

```
ModeFE_Package (ModeFE_Label *,
                char * ,
                ModeFE_NameList * ,
                ModeFE_AttributeAndPropertiesList *,
                ModeFE_LabelAssociationList *,
                ModeFE_LabelAssociationList *,
                ModeFE_LabelAssociationList *);
```

Basic constructor of the ModeFE_Package class. The parameters of this method are:

1. the package label,
2. its registration identifier, `NULL` if none,
3. the list of behaviour labels associated with the package,
4. the list of attributes and their properties,
5. the list of actions and the associated parameters,
6. the list of notifications and the associated parameters.

```
ModeFE_Package ();
```

Default constructor for a ModeFE_Package class. Creates an empty instance of a package specification.

```
ModeFE_Package (const ModeFE_Package&);
```

Copy constructor of the ModeFE_Package class. Returns no element in the current version of the library.

```
~ModeFE_Package ();
```

Destructor of the ModeFE_Package. Deletes all contained elements.

```
ModeFE_Label* GetLabel();
```

Returns a pointer to a copy of the package type label. If no label is associated, returns a pointer to an empty ModeFE_Label object.

```
void SetLabel(ModeFE_Label* );
```

Assigns a new value to the package type label.

```
char* GetRegistration();
```

Returns a pointer to a copy of the package type registration identifier. The identifier is provided as a string and does not contain the "{" and "}" brackets.

```
void SetRegistration(char*);
```

Assigns a new registration identifier to the package specification. The registration identifier should not include the start and stop brackets ("{","}").

```
ModeFE_NameList* GetBehaviours();
```

Returns the labels of all behaviour definitions associated with the package type. If no behaviour is associated, then this methods returns an empty list.

int AddBehaviour(ModeFE_Label *);

Adds one behaviour label to the head of the list of behaviours associated with the package type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AppendBehaviour(ModeFE_Label *);

Adds one behaviour label to the end of the list of behaviours associated with the package type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int RemoveBehaviour(ModeFE_Label *);

Removes the first occurence of a behaviour label in the list of behaviour labels associated with the package type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplaceBehaviour(ModeFE_Label *, ModeFE_Label*);

Replaces the first occurence of the first parameter in the list of behaviour labels associated with the package type with the label given as a second parameter. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

void ClearBehaviours();

Removes all behaviour labels from the list of behaviour labels associated with the package type.

ModeFE_AttributeAndPropertiesList * GetAttributesAndProperties();

Returns the list of all attributes and the associated properties defined in the package type. If no attribute is associated, then this methods returns an empty list.

int AddAttributeAndProperties(ModeFE_AttributeAndProperties*);

Adds one attribute and its properties to the front of the list of all attributes and the associated properties defined in the package type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AppendAttributeAndProperties(ModeFE_AttributeAndProperties*);

Adds one attribute and its properties to the end of the list of all attributes and the associated properties defined in the package type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int RemoveAttributeAndProperties(ModeFE_AttributeAndProperties*);

Removes the first occurence of an attribute and its properties from the list of attribute and properties associated defined within the package. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplaceAttributeAndProperties(ModeFE_AttributeAndProperties *, ModeFE_AttributeAndProperties*);

Replaces the first occurence of an attribute and its properties that is equal to the first parameter with the second parameter in the list of attribute and properties associated defined within the package. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

void ClearAttributeAndProperties();

Removes all attribute and properties defined in the package.

ModeFE_LabelAssociationList * GetAttributeGroups();

Returns the list of attribute groups and the associated attributes defined in the package.

int AddAttributeGroup(ModeFE_LabelAssociationElement*);

Adds one attribute group and additional attributes to the head of list of attribute groups defined in this package. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AppendAttributeGroup(ModeFE_LabelAssociationElement*);

Adds one attribute group and additional attributes to the end of list of attribute groups defined in this package. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int RemoveAttributeGroup(ModeFE_LabelAssociationElement *);

Removes the first occurence of an attribute group and associated attributes in the list of attribute groups defined in the package. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplaceAttributeGroup(ModeFE_LabelAssociationElement*, ModeFE_LabelAssociationElement *);

Replaces the first occurence of the first parameter in the list of attribute group defined in the package with the attribute group given as a second parameter. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

void ClearAttributeGroups();

Deletes all attribute groups defined in the package.

ModeFE_LabelAssociationList* GetActionAndParameters();

Returns a pointer to a copy of all action labels and associated parameter labels defined in the package.

int AddActionAndParameter(ModeFE_LabelAssociationElement*);

Adds one action and the associated parameters to the head of the list of actions and parameters defined in the package. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AppendActionAndParameter(ModeFE_LabelAssociationElement*);

Adds one action and the associated parameters to the end of the list of actions and parameters defined in the package. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int RemoveActionAndParameter(ModeFE_LabelAssociationElement *);

Removes the first occurence of an action and the corresponding parameters in the list of actions and parameters associated with the package type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplaceActionAndParameter(ModeFE_LabelAssociationElement*, ModeFE_LabelAssociationElement *);

Replaces the first occurence of the first parameter in the list of actions and parameters associated with the package with the action and parameter specification given as a second parameter. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

void ClearActionAndParameter();

Removes all action and parameters from the package specification.

ModeFE_LabelAssociationList* GetNotificationAndParameterList();

Returns a pointer to a copy of all notification labels and associated parameter labels defined in the package.

int AddNotificationAndParameter(ModeFE_LabelAssociationElement*);

Adds one notification and the associated parameters to the head of the list of notifications and parameters defined in the package. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AppendNotificationAndParameter(ModeFE_LabelAssociationElement*);

Adds one notification and the associated parameters to the end of the list of notifications and parameters defined in the package. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int RemoveNotificationAndParameter(ModeFE_LabelAssociationElement *);

Removes the first occurence of a notification and the corresponding parameters in the list of notification and parameters associated with the package type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplaceNotificationAndParameter(ModeFE_LabelAssociationElement*,
ModeFE_LabelAssociationElement *);

Replaces the first occurence of the first parameter in the list of notifications and parameters associated with the package with the notification and parameter specification given as a second parameter. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

void ClearNotificationAndParameter();

Removes all notifications and associated parameters from the package specification.

int IsEqual(ModeFE_Package*);

Returns `TRUE` if the package given in parameter is equal (has the same label) to the current package. Returns `FALSE` otherwise.

ModeFE_Package* Dupplicate();

Returns a pointer to a copy of the current object. The invoker is then responsible for deleting this pointer and its content.

void Print(ofstream *);

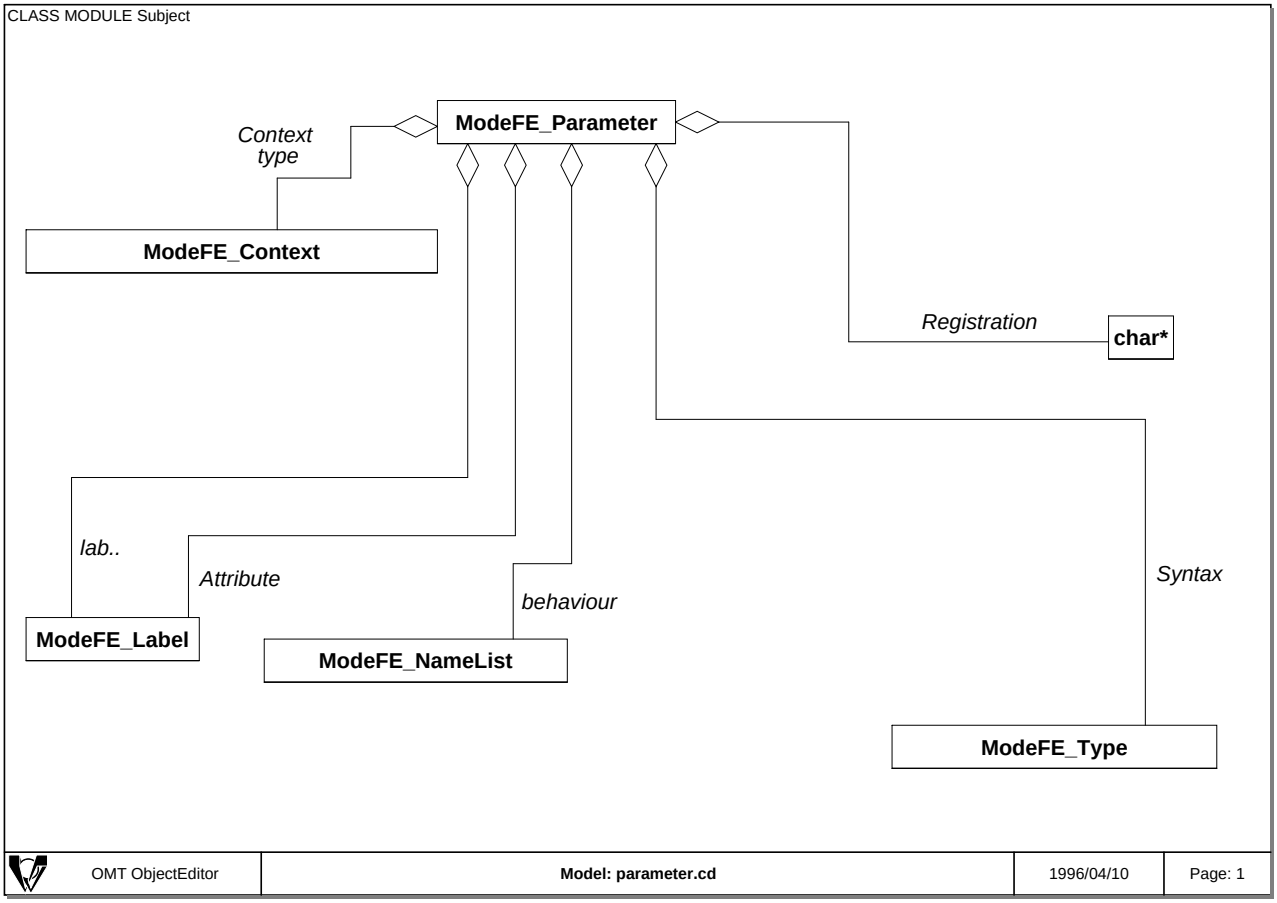
Prints in an ASCII format to the stream, the GDMO specification that corresponds tot this package type specification.

A.1.18 The ModeFE_Parameter class

Purpose

The ModeFE_Parameter class sotres information related to the specification of a GDMO parameter type. This information is the parameters label, its registration identifier, its usage context, its ASN.1 syntax if any, the label of the attribute from which it is derived if any, and a list of behaviour labels that correspond to behaviour definitions associated with the parameter.

Architecture



Available methods

```
ModeFE_Parameter ( ModeFE_Label * ,
                   char * ,
                   ModeFE_Context * ,
                   ModeFE_Type ,
                   ModeFE_Label * ,
                   ModeFE_NameList * );
```

basic constructor of the ModeFE_Parameter class. The parameters of this method are:

1. the label associated with the parameter,
2. its registration identifier, `NULL` if none),
3. the context associated with the parameter,
4. its ASN.1 type, if any (`NULL` if none),
5. the associated attribute, if any (`NULL` or an empty label if none),

6. the list of associated behaviour labels (`NULL` or an empty NameList if none).

```
ModeFE_Parameter ();
```

Default constructor for the ModeFE_Parameter class. Creates an empty instance.

```
ModeFE_Parameter (const ModeFE_Parameter&);
```

Copy constructor of the ModeFE_Parameter class. Returns no element in the current version of the library.

```
~ModeFE_Parameter () ;
```

Destructor of the ModeFE_Parameter class. Deletes all contained elements.

```
ModeFE_Label* GetLabel();
```

Returns a pointer to a copy of the parameter type label. If no label is associated, returns a pointer to an empty ModeFE_Label object.

```
int SetLabel(ModeFE_Label*);
```

Assigns a new value to the parameter type label. Returns `OK` if the operation was successful. Returns `ERROR` otherwise.

```
ModeFE_Label* GetAttribute();
```

Returns a pointer to a copy of the label which identifies the attribute associated with the parameter. If no label is associated, returns a pointer to an empty ModeFE_Label object.

```
int SetAttribute(ModeFE_Label*);
```

Assigns a new value to the label which identifies the attribute associated with the parameter. If one whishes to associated no attribute with the parameter, the value of the methods parameter can be either `NULL` or an empty label. The method returns `OK` if the operation was successful. Returns `ERROR` otherwise.

```
char* GetRegistration();
```

Returns a pointer to a copy of the parameter type registration identifier. The identifier is provided as a string and does not contain the "{" and "}" brackets. If no registration identifier exists, the method returns `NULL`.

```
void SetRegistration(char*);
```

Assigns a new registration identifier to the parameter specification. The registration identifier should not include the start and stop brackets ("{","}"). The parameter of the method can be `NULL` and thus no registration identifier will be associated with the parameter type.

```
ModeFE_NameList* GetBehaviours();
```

Returns a pointer to a copy of the list of labels of all behaviour definitions associated with the parameter type. If no behaviour is associated, then this methods returns an empty list.

```
int AddBehaviour(ModeFE_Label *);
```

Adds one behaviour label to the head of the list of behaviours associated with the paarmeter type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int AppendBehaviour(ModeFE_Label *);
```

Adds one behaviour label to the end of the list of behaviours associated with the paarmeter type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int RemoveBehaviour(ModeFE_Label *);
```

Removes the first occurence of a behaviour label in the list of behaviour labels associated with the parameter type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int ReplaceBehaviour(ModeFE_Label *, ModeFE_Label*);
```

Replaces the first occurence of the first parameter in the list of behaviour labels associated with the parameter type with the label given as a second parameter. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
void ClearBehaviours();
```

Removes all labels from the list of behaviour labels associated with this parameter definition.

```
ModeFE_Type GetSyntax();
```

Returns a pointer to a copy of the ASN.1 type that corresponds to the parameters type. If no type is associated with the parameter, the method returns `NULL`.

```
int SetSyntax(ModeFE_Type);
```

Assigns a new value to the ASN.1 type that corresponds to the parameters type. If no type is associated with the parameter, the methods parameter must be equal to `NULL`. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
ModeFE_Context * GetContext();
```

Returns a pointer to a copy of the context associated with the parameter.

```
int SetContext(ModeFE_Context*);
```

Assigns a new value to the context associated with the parameter. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int IsEqual(ModeFE_Parameter*);
```

Returns `TRUE` if the parameter specification given in parameter is equal to the current one. Returns `FALSE` otherwise. The equality test is made only on the parameter specification label in this version of the library.

```
ModeFE_Parameter* Dupplicate();
```

Returns a pointer to a copy of the current object. The invoker is then responsible for deleting this pointer and its content.

```
void Print ( ofstream * ) ;
```

Prints in an ASCII format to the stream, the GDMO specification that corresponds to this parameter type specification.

The ModeFE_RelationshipClass class

Purpose

The ModeFE_RelationshipClass stores information related to the specificaiton of a GRM relationship class. The library stores one instance of a ModeFE_RelationshipClass per GRM relationship class. The information associated with a relationship class specification is the relationship class label, its registration identifier, the list of super-classes from which the relationship class is derived, the lis of behaviour labels which indentify behaviours assocaited with the relationship class, the list of relaitonship operations supported by the relationship class, the list of attributes that qualify the relationship and the list of roles defined within the relaitonship.

Available methods

```
ModeFE_RelationshipClass (ModeFE_Label *,
                          ModeFE_NameList *,
                          ModeFE_NameList *,
                          ModeFE_SupportedList *,
                          ModeFE_NameList *,
                          ModeFE_RoleSpecificationList *,
                          char *);
```

Basic constructor for the ModeFE_RelationshipClass class. The parameters of this method are:

1. the label of the relationship class,
2. the list of relationship classes from which this class is derived,
3. the list of behaviour labels associated with the relationship class,
4. the list of supported relationship operations,
5. the list of labels describing qualifier attributes,
6. the list of role specifications defined in the relationship,
7. the registration identifier associated with the relationship class.

```
ModeFE_RelationshipClass ();
```

Default constructor of the ModeFE_RelationshipClass class. Creates an empty instance of a relationship class specification.

```
ModeFE_RelationshipClass (const ModeFE_RelationshipClass&);
```

Copy constructor of the ModeFE_RelationshipClass class. Returns no element in the current version of the library.

```
~ModeFE_RelationshipClass ();
```

Destructor of the ModeFE_RelationshipClass class. Deletes all contained elements.

```
ModeFE_Label* GetLabel();
```

Returns a pointer to a copy of the relationship class type label. If no label is associated, returns a pointer to an empty ModeFE_Label object.

void SetLabel(ModeFE_Label*);

Assigns a new value to the relationship class type label.

char* GetRegistration();

Returns a pointer to a copy of the relationship class type registration identifier. The identifier is provided as a string and does not contain the "{" and "}" brackets.

void SetRegistration(**char***);

Assigns a new registration identifier to the relationship class specification. The registration identifier should not include the start and stop brackets ("{","}").

ModeFE_NameList* GetParentClasses();

Returns a pointer to a copy of the list of the labels of the super classes from which this relationship class is derived. If no super-class exists the method returns a empty list.

int AddParentClass(ModeFE_Label *);

Adds one label to the head of the list of super-classes associated with this relationship class. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AppendParentClass(ModeFE_Label *);

Adds one label to the end of the list of super-classes associated with this relationship class. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int RemoveParentClass(ModeFE_Label *);

Removes one label from the list of super-classes associated with this relationship class. This method returns `OK` if the operation completed successfully. It returns `ERROR` otherwise.

int ReplaceParentClass(ModeFE_Label *, ModeFE_Label*);

Replaces the first occurence of the first parameter in the list of super classes labels associated with the relationship type with the label given as a second parameter. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

void ClearParentList();

Removes all super-classes labels from the list of parent classes labels associated with the relationship class type.

ModeFE_NameList* GetBehaviours();

Returns the labels of all behaviour definitions associated with the relationship class type. If no behaviour is associated, then this methods returns an empty list.

int AddBehaviour(ModeFE_Label *);

Adds one behaviour label to the head of the list of behaviours associated with the relationship class type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AppendBehaviour(ModeFE_Label *);

Adds one behaviour label to the end of the list of behaviours associated with the relationship class type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int RemoveBehaviour(ModeFE_Label *);

Removes the first occurence of a behaviour label in the list of behaviour labels associated with the relationship class type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplaceBehaviour(ModeFE_Label *, ModeFE_Label*);

Replaces the first occurence of the first parameter in the list of behaviour labels associated with the relationship class type with the label given as a second parameter. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

void ClearBehaviours();

Removes all behaviour labels from the list of behaviour labels associated with the relationship class type.

ModeFE_SupportedList* GetSupport();

Returns a pointer to a copy of the list of supported relationship operations associated with this relationship class. If no relationship operation exists, the method returns an empty list.

int AddSupport(ModeFE_Supported *);

Adds one relationship operation to the head of the list of relationship operations defined for the relationship class. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AppendSupport(ModeFE_Supported *);

Adds one relationship operation to the end of the list of relationship operations defined for the relationship class. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int RemoveSupport(ModeFE_Supported *);

Removes the first occurence of the supported relationship operation from the list of relationship operations associated with the MO type. The selection is made only on the relaitonship operaion name. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplaceSupport(ModeFE_Supported *, ModeFE_Supported*);

Replaces the first occurence of the relationship operation given in the first parameter in the list of relationship operations associated with the relationship class type with the second parameter. The selection is made only on the relationship operation name. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

void ClearSupport();

Removes all relationship operations defined in this relationship class.

ModeFE_NameList* GetQualifiers();

Returns a pointer to a copy of all attributes which qualify the relationship. If no qualifier exists, the method returns an empty list.

int AddQualifier(ModeFE_Label *);

Adds one attribute label to the head of the list of attributes labels which qualify the relationship class. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AppendQualifier(ModeFE_Label *);

Adds one attribute label to the end of the list of attributes labels which qualify the relationship class. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int RemoveQualifier(ModeFE_Label *);

Removes the first occurence of the attribute label given in the methods parameter from the list of attributes which qualify the relationship associated with the relationship class type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplaceQualifier(ModeFE_Label *, ModeFE_Label*);

Replaces the first occurence of the attribute label given in the first parameter in the list of attributes which qualify the relationship asociated with the relationship class type with the second parameter. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

void ClearQualifiers();

Removes the labels of all qualifying attributes associated with the relationship class.

ModeFE_RoleSpecificationList* GetRoles();

Returns a pointer to a copy of all roles defined in the context of this relationship. If no role is defined, the method returns an empty list.

int AddRole(ModeFE_RoleSpecification *);

Adds one role to the head of the list of roles defined within the relationship class. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AppendRole(ModeFE_RoleSpecification *);

Adds one role to the end of the list of roles defined within the relationship class. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int RemoveRole(ModeFE_RoleSpecification *);

Removes the first occurence of a role from the list of roles associated with the relationship class type. The selection is mande only on the role name, in this version of the library. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplaceRole(ModeFE_RoleSpecification *, ModeFE_RoleSpecification*);

Replaces the first occurence of the first parameter in the list of roles associated with the relationship class type with the role given as a second parameter. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

void ClearRoles();

Removes all roles from the relationship class specification.

int IsEqual(ModeFE_RelationshipClass*);

Returns `TRUE` if the relationship class given in parameter is equal to the current one. Returns `FALSE` otherwise. The equality test is made only on the relationship class label in this version of the library.

ModeFE_RelationshipClass* Dupplicate();

Returns a pointer to a copy of the current object. The invoker is then responsible for deleting this pointer and its content.

void Print(ofstream *) ;

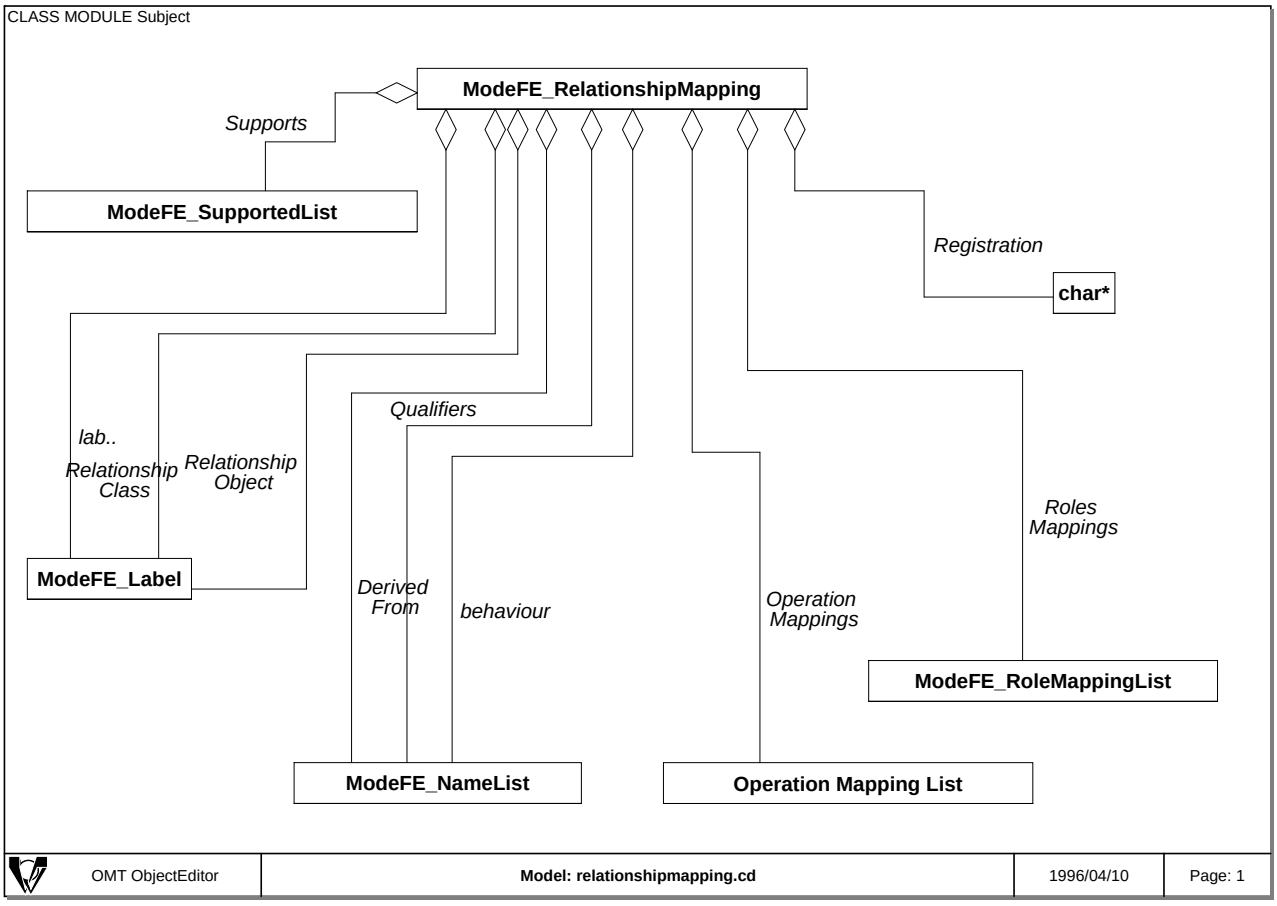
Prints in an ASCII format to the stream, the GRM specification that corresponds to this relationship class type specification.

The ModeFE_RelationshipMapping class

Purpose

The ModeFE_RelationshipMapping class is used to store information contained in a GRM relationship mapping specification. This information covers the mapping label, its registration identifier, the label of the corresponding relationship class, the list of attributes which qualify the relationship, the label of the relationship object, if any, the list of behaviours associated with this mapping, the list of role mappings as well as the list of operation mappings.

Architecture



Available methods

```
ModeFE_RelationshipMapping (ModeFE_Label * ,
                            ModeFE_Label * ,
                            ModeFE_NameList * ,
                            ModeFE_Label * ,
                            ModeFE_NameList * ,
                            ModeFE_RoleMappingList * ,
                            ModeFE_OperationMappingList * ,
                            char * );
```

Basic constructor for the ModeFE_RelationshipMapping class. The parameters are:

1. the label of the mapping,
2. the label of the corresponding relationship class,
3. the list of behaviour labels associated with this mapping,
4. the label of the associated relationship object, if any,
5. the list of attributes which qualify the mapping in the relationship object,
6. the list of role mappings,
7. the list of operation mappings,
8. the registration identifier of the relationship mapping.

```
ModeFE_RelationshipMapping();
```

Default constructor for the ModeFE_RelationshipMapping class. Creates an empty instance of a relationship mapping.

```
ModeFE_RelationshipMapping(const ModeFE_RelationshipMapping&);
```

Copy constructor of the ModeFE_RelationshipMapping class. Returns no element in the current version of the library.

```
~ModeFE_RelationshipMapping();
```

Destructor of the ModeFE_RelationshipMapping class. Deletes all contained elements.

```
ModeFE_Label* GetLabel();
```

Returns a pointer to a copy of the relationship mapping type label. If no label is associated, returns a pointer to an empty ModeFE_Label object.

```
void SetLabel(ModeFE_Label* );
```

Assigns a new value to the relationship mapping type label.

```
char* GetRegistration();
```

Returns a pointer to a copy of the relationship mapping type registration identifier. The identifier is provided as a string and does not contain the "{" and "}" brackets.

```
void SetRegistration(char*);
```

Assigns a new registration identifier to the relationship mapping specification. The registration identifier should not include the start and stop brackets ("{","}").

```
ModeFE_Label* GetRelationshipObject();
```

Returns a pointer to a copy of the label which refers to the relationship object associated to this relationship mapping. If no relationship object exists, then the method returns an empty label.

```
int SetRelationshipObject(ModeFE_Label* );
```

Assigns a new value to the label of the relationship object associated with this relationship mapping. If one wants to associated no relationship object with the mapping the argument of the method can be either `NULL` or an empty label.

```
ModeFE_Label* GetRelationshipClass();
```

Returns a pointer to a copy of the label which refers to the relationship class associated to this relationship mapping.

```
void SetRelationshipClass(ModeFE_Label* );
```

Assigns a new value to the label of the relationship class with which with this relationship mapping is associated.

```
ModeFE_NameList* GetBehaviours();
```

Returns acopy of the labels of all behaviour definitions associated with the relationship mapping type. If no behaviour is associated, then this methods returns an empty list.

```
int AddBehaviour(ModeFE_Label *);
```

Adds one behaviour label to the head of the list of behaviours associated with the relationship mapping type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int AppendBehaviour(ModeFE_Label *);
```

Adds one behaviour label to the end of the list of behaviours associated with the relationship mapping type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int RemoveBehaviour(ModeFE_Label *);
```

Removes the first occurence of a behaviour label in the list of behaviour labels associated with the relationship mapping type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int ReplaceBehaviour(ModeFE_Label *, ModeFE_Label*);
```

Replaces the first occurence of the first parameter in the list of behaviour labels associated with the relationship mapping type with the label given as a second parameter. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
void ClearBehaviours();
```

Removes all behaviour labels from the list of behaviour labels associated with the relationship mapping type.

```
ModeFE_NameList* GetQualifies();
```

Returns the list of attributes which qualify the mapping. If no attribute exists, the method returns an empty list.

```
int AddQualifies(ModeFE_Label *);
```

Adds one attribute label to the head of the list of qualifier attributes associated with the relaitonship mapping type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int AppendQualifies(ModeFE_Label *);
```

Adds one attribute label to the end of the list of qualifier attributes associated with the relaitonship mapping type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int RemoveQualifies(ModeFE_Label *);
```

Removes the first occurence of the attribute label which is equal to the methods parameter from the list of qualifier attributes associated with the relaitonship mapping type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int ReplaceQualifies(ModeFE_Label *, ModeFE_Label*);
```

Replaces the first occurence of the first parameter in the list of qualifier attribtute labels associated with the relationship mapping type with the label given as a second parameter. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
void ClearQualifies();
```

Removes all qualifier labels from the list of qualifiers associated with this relationship mapping.

```
ModeFE_RoleMappingList* GetRoleMappings();
```

Returns a pointer to a copy of all role mappings defined in this relationship mapping. If no role mapping exists, the method returns an empty list.

```
int AddRoleMapping(ModeFE_RoleMapping *);
```

Adds one role mapping to the head of the list of role mappings defined in this relationship mapping. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int AppendRoleMapping(ModeFE_RoleMapping *);
```

Adds one role mapping to the end of the list of role mappings defined in this relationship mapping. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int RemoveRoleMapping(ModeFE_RoleMapping *);
```

Removes the first occurence of a role mapping that is equal to the role mapping given as a parameter to the method from the list of role mappings associated with the relationship mapping. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int ReplaceRoleMapping(ModeFE_RoleMapping *, ModeFE_RoleMapping*);
```

Replaces the first occurence of a role mapping that is equal to the role mapping given as the first parameter to the method with the second parameter in the list of role mappings associated with the relationship mapping. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

void ClearRoleMappings();

Removes all role mappings defined in this relationship mapping.

ModeFE_OperationMappingList* GetOperationMappings();

Returns a pointer to a copy of all operation mappings defined in this relationship mapping. If no operation mapping exists, the method returns an empty list.

int AddOperationMapping(ModeFE_OperationMapping *);

Adds one operation mapping to the head of the list of operation mappings defined in this relationship mapping. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AppendOperationMapping(ModeFE_OperationMapping *);

Adds one operation mapping to the end of the list of operation mappings defined in this relationship mapping. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int RemoveOperationMapping(ModeFE_OperationMapping *);

Removes the first occurence of an operation mapping that is equal to the operation mapping given as a parameter to the method from the list of operation mappings associated with the relationship mapping. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplaceOperationMapping(ModeFE_OperationMapping *, ModeFE_OperationMapping*);

Replaces the first occurence of an operation mapping that is equal to the operation mapping given as the first parameter to the method with the second parameter in the list of operation mappings associated with the relationship mapping. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

void ClearOperationMappings();

Removes all operation mappings defined in this relationship mapping.

int IsEqual(ModeFE_RelationshipMapping*);

Returns `TRUE` if the mapping given as a parameter to the method has the same label as the current one. Returns `FALSE` otherwise.

ModeFE_RelationshipMapping* Dupplicate();

Returns a pointer to a copy of the current object. The invoker is then responsible for deleting this pointer and its content.

void Print(ofstream *);

Prints in an ASCII format to the stream, the GRM specification that corresponds to this relationship mapping type specification.

The ModeFE_RelationshipOperation class

Purpose

The ModeFE_RelationshipOperation class stores information related to a relationship operation. This information covers the type of the relationship operation, its name (optional), and the name of the role to which this operation applies.

Available methods

```
ModeFE_RelationshipOperation(ModeFE_REL_OP ,
                             char *,
                             char *);
```

Basic constructor of the ModeFE_RelationshipOperation class. The parameters are:

1. the type of the relationship operation. Possible values are:
   - `OP_ESTABLISH_`
   - `OP_TERMINATE_`
   - `OP_QUERY_`
   - `OP_NOTIFY_`
   - `OP_USER_DEFINED_`
   - `OP_BIND_`
   - `OP_UNBIND_`
2. the name of the operation (`NULL` if none).
3. the name of the role to which this operation applies (`NULL` if none).

```
ModeFE_RelationshipOperation();
```

Default constructor of the ModeFE_RelationshipOperation class. Creates an empty relaitonship operation instance.

```
ModeFE_RelationshipOperation(const ModeFE_RelationshipOperation&);
```

Copy constructor of the ModeFE_RelationshipOperation class. Returns no element in the current version of the library.

```
~ModeFE_RelationshipOperation();
```

Destructor of the ModeFE_RelationshipOperation class. Deletes all contained elements.

```
ModeFE_REL_OP GetOperationType();
```

Returns the type of the relationship operation. Possible values for the response are:

- `OP_ESTABLISH_`
- `OP_TERMINATE_`
- `OP_QUERY_`
- `OP_NOTIFY_`
- `OP_USER_DEFINED_`

- OP_BIND_
- OP_UNBIND_

int SetOperationType(ModeFE_REL_OP);

Assigns a new type to the relationship operation. Possible values for the parameter are:

- OP_ESTABLISH_
- OP_TERMINATE_
- OP_QUERY_
- OP_NOTIFY_
- OP_USER_DEFINED_
- OP_BIND_
- OP_UNBIND_

This method returns OK if the operation was successful. It returns ERROR otherwise.

char* GetOperationName();

Returns a pointer to a copy of the operations name. If no name is specified for the action, the method returns NULL

int SetOperationName(**char** *);

Assigns a new value to the name associated with the operation. If no name has to be associated with the operation, the parameter must be NULL. This method returns OK if the operation was successful. It returns ERROR otherwise.

char* GetRoleName();

Returns a pointer to a copy of the role name to which the operation applies. If no name is specified for the operation, the method returns NULL

int SetRoleName(**char** *);

Assigns a new value to the role associated with the operation. If no role has to be associated with the operation, the parameter must be NULL. This method returns OK if the operation was successful. It returns ERROR otherwise.

ModeFE_RelationshipOperation* Dupplicate();

Returns a pointer to a copy of the current object. The invoker is then responsible for deleting this pointer and its content.

void Print(ofstream *);

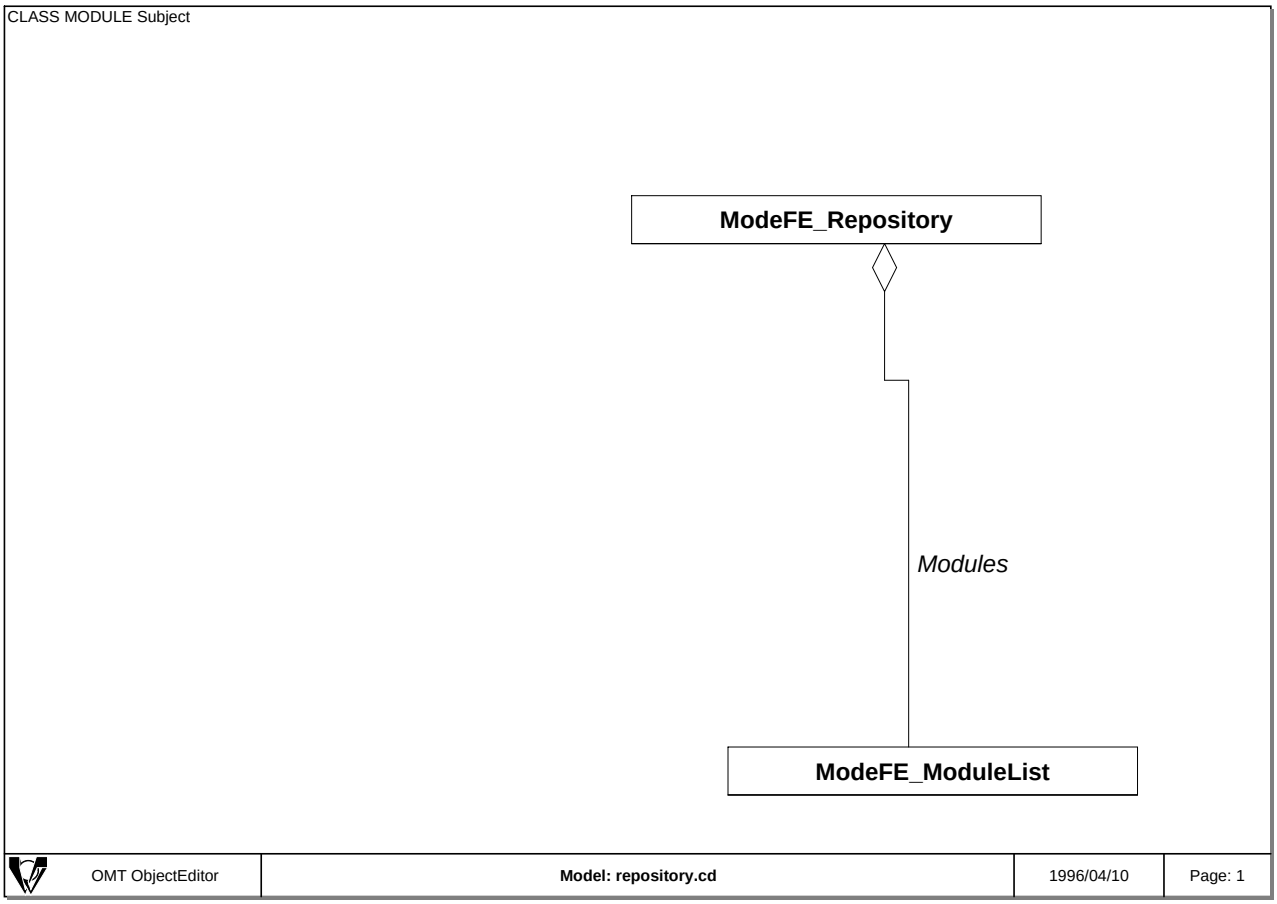
Prints in an ASCII format to the stream, the GRM specification that corresponds to this relaitonship operation type specification.

The ModeFE_Repository class

Purpose

The ModeFE_Repository stores a list of GDMO and GRM modules.

Architecture



Available methods

ModeFE_Repository(ModeFE_ModuleList*);

Basic constructor of the ModeFE_Repository class. The methdo takes a list of specification modules in parameter.

ModeFE_Repository();

Default constructor for the ModeFE_Repository class. Creates a repository with no contained module specifications.

ModeFE_Repository(**const** ModeFE_Repository&);

Copy constructor of the ModeFE_Repository class. Returns no element in the current version of the library.

ModeFE_Repository(**const** ModeFE_Repository&);

Destructor of the ModeFE_Repository class. Deletes all contained elements.

ModeFE_ModuleList * GetModules();

Returns a pointer to a copy of all modules defined in the repository. If no module is present, the method returns an empty list.

void SetModules(ModeFE_ModuleList*);

Assigns the set of module specifications defined in the repository to the list given in parameter.

int AddModule(ModeFE_Module*);

Adds one module to the head of the list of modules defined in the repository. The method returns `OK` if the operation was successful, `ERROR` otherwise.

int AppendModule(ModeFE_Module*);

Adds one module to the end of the list of modules defined in the repository. The method returns `OK` if the operation was successful, `ERROR` otherwise.

int RemoveModule(ModeFE_Module *);

Removes the first occurence of a module which has the same identifier as the module given in the parameter of the method from the list of modules. The method returns `OK` if the operation was successful, `ERROR` otherwise.

int ReplaceModule(ModeFE_Module *, ModeFE_Module *);

Replaces the first occurence of the first parameter in the list of modules with the module given as a second parameter. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

void ClearModules();

Removes all modules from the list of modules defined in the repository.

ModeFE_Repository* Dupplicate();

Returns a pointer to a copy of the repository.

void Print (ofstream *) ;

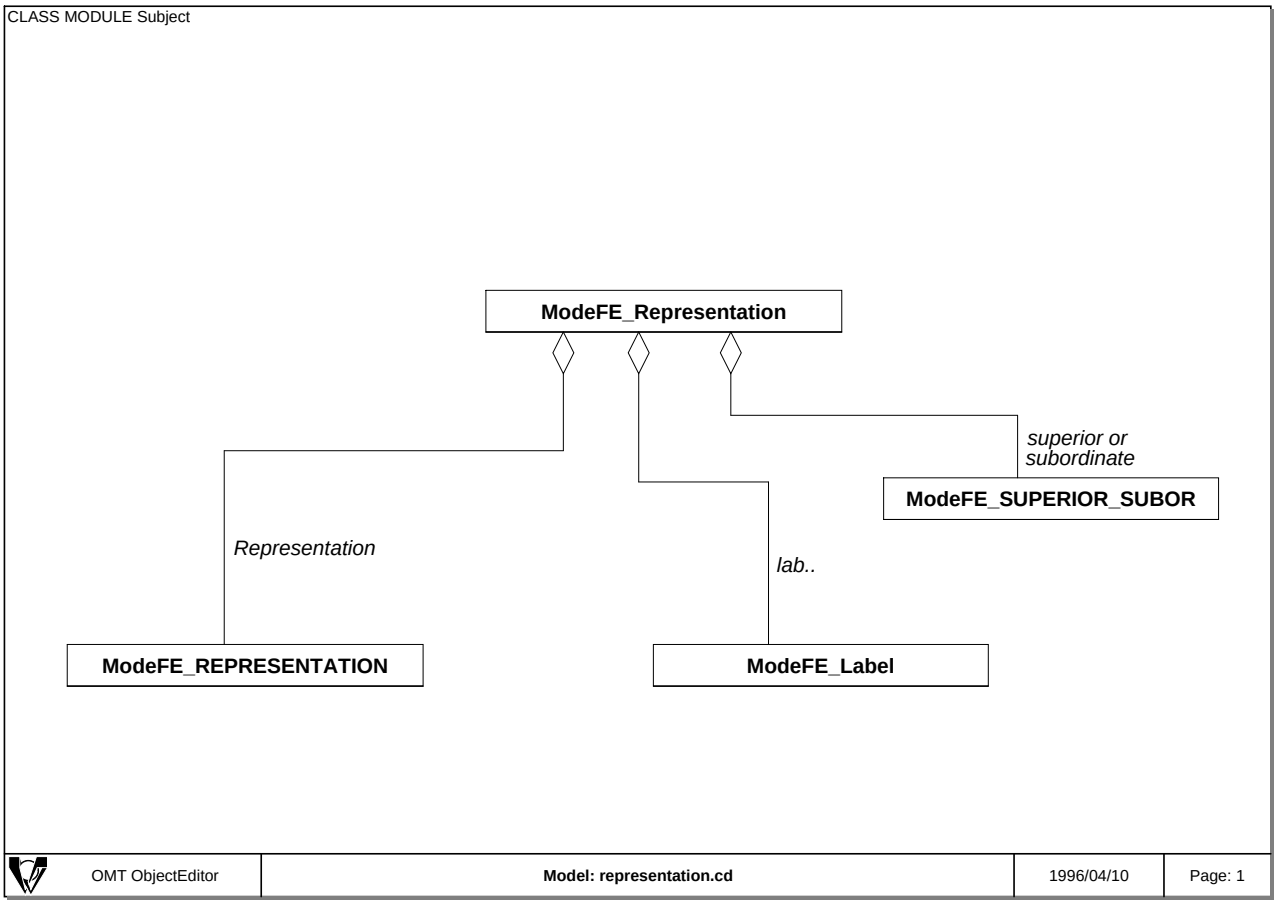
Prints in an ASCII format to the stream, the GDMO and GRM specifications of all modules defined in the repository.

The ModeFE_Representation class

Purpose

The ModeFE_Representation class is used to store information related to the representation of a role mapping in a GRM relationship mapping specification. The information available in a represnetation is the type of the representation, the label of an associated attribute if any and/or the label of an associated Name-Binding if any.

Architecture



Available methods

```
ModeFE_Representation(ModeFE_REPRESENTATION ,
                      ModeFE_Label * ,
                      ModeFE_SUPERIOR_SUBOR );
```

Basic constructor of the ModeFE_Representation class. The parameters of this method are:

1. the representation type. Possible values are:
   - `R_NAMING_`
   - `R_ATTRIBUTE_`
   - `R_OBJECT_`
   - `R_OPERATION_`
   - `R_NO_REP_`
2. the label of either the name-binding associated with the representation or the attribute associated with the representation (`NULL` or an empty label if no label has to be associated).
3. an indicator which states that either subordinate or superior class is used in conjunction with a Name-Binding. Possible values for this parameter are:
   - `NONE_`
   - `SUPERIOR_`
   - `SUBORDINATE_`

ModeFE_Representation();

Default constructor for the ModeFE_Representation class. Creates an empty representation instance.

ModeFE_Representation(**const** ModeFE_Representation&);

Copy constructor of the ModeFE_Representation class. Returns no element in the current version of the library.

~ModeFE_Representation();

Destructor of the ModeFE_Representation class. Deletes all contained elements.

ModeFE_REPRESENTATION GetRepresentationType();

Returns the representaion types. Possible values for the response are:

- `R_NAMING_`
- `R_ATTRIBUTE_`
- `R_OBJECT_`
- `R_OPERATION_`
- `R_NO_REP_`

int SetRepresentationType(ModeFE_REPRESENTATION);

Assigns a new representaion type to the current representation. Possible values for the parameter are:

- `R_NAMING_`
- `R_ATTRIBUTE_`
- `R_OBJECT_`
- `R_OPERATION_`
- `R_NO_REP_`

The method returns `OK` if the operation was successful. Returns `ERROR` otherwise.

ModeFE_Label* GetLabel();

Returns a pointer to a copy of the label associated with the representation. Depending on the representation type, this label can be a Name-Binding label, an attribute label or an empty label.

int SetLabel(ModeFE_Label*);

Assigns a new label to the representation. Depending on the representation type, this label can be a Name-Binding label, an attribute label, an empty label or the `NULL` value. The method returns `OK` if the operation was successful. Returns `ERROR` otherwise.

ModeFE_SUPERIOR_SUBOR GetSuborSup();

Returns the statement on subordinate or superior object choice in the case of a Name-Binding based representation. Possible values for the response are:

- NONE_
- SUPERIOR_
- SUBORDINATE_

int SetSuborSup(ModeFE_SUPERIOR_SUBOR);

Assigns a new statement on subordinate or superior object choice in the case of a Name-Binding based representation. Possible values for the parameter are:

- NONE_
- SUPERIOR_
- SUBORDINATE_

The method returns OK if the operation was successful, ERROR otherwise.

ModeFE_Representation* Dupplicate();

Returns a pointer to a copy of the current object. The invoker is then responsible for deleting this pointer and its content.

void Print(ofstream *);

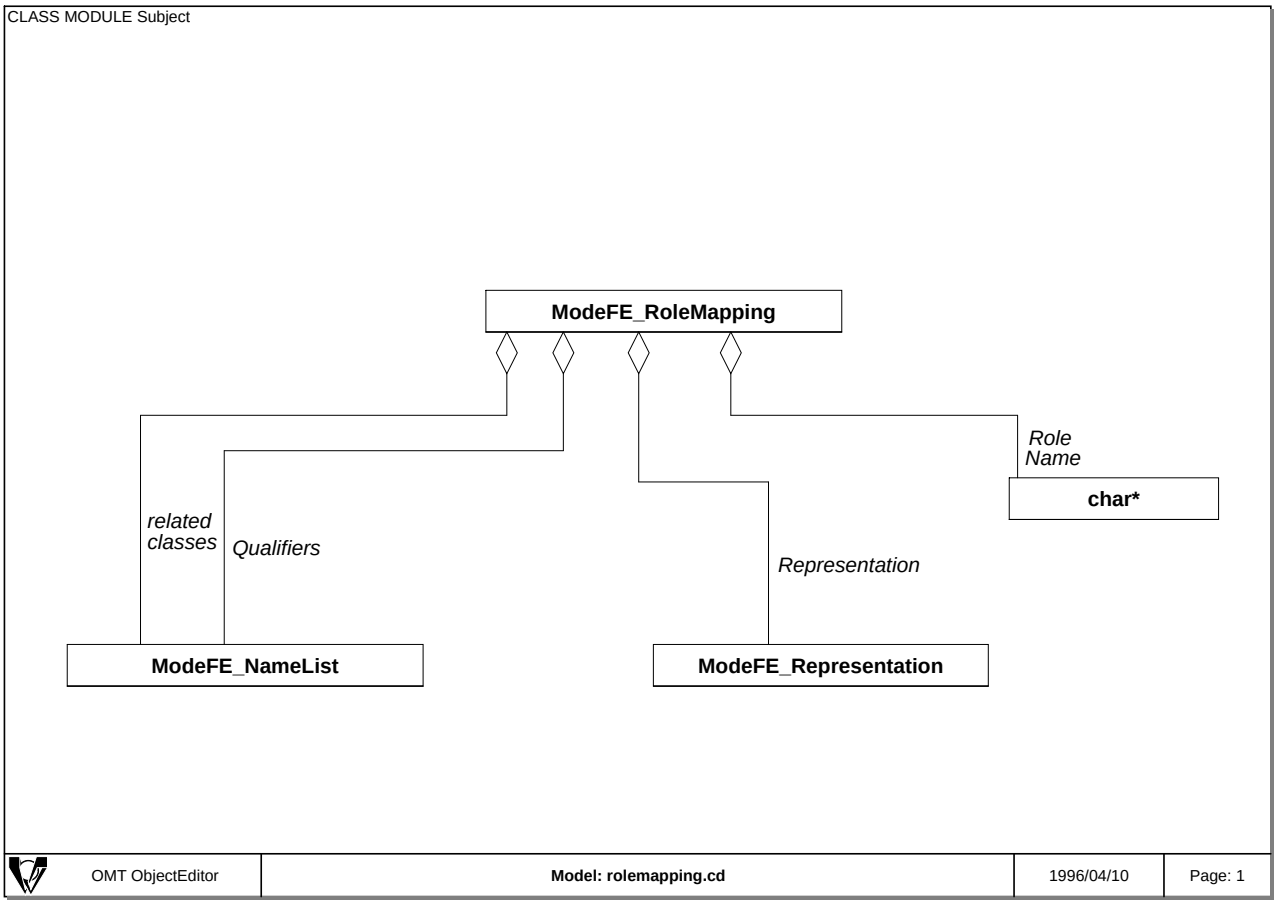
Prints in an ASCII format to the stream, the GDMO specification that corresponds to this representation type specification.

The ModeFE_RoleMapping class

Purpose

The ModeFE_RoleMapping stores information related to a role mapping specification. This information is the name of the role, the list of related classes (candidate managed object classes), the associated representation and the list of qualifying attributes.

Architecture



Available methods

```
ModeFE_RoleMapping(char * a,
                   ModeFE_NameList * b,
                   ModeFE_Representation* c,
                   ModeFE_NameList * g);
```

Basic constructor of the ModeFE_RoleMapping class. The parameters of the method are:

1. the name of the role
2. the list of candidate managed object classes for the role,
3. the representation of the role
4. the list of attributes which qualify the role.

```
ModeFE_RoleMapping();
```

Default constructor of the ModeFE_RoleMapping class. Creates an empty role mapping.

```
ModeFE_RoleMapping(const ModeFE_RoleMapping&);
```

Copy constructor of the ModeFE_RoleMapping class. Returns no element in the current version of the library.

~ModeFE_RoleMapping();

Destructor of the ModeFE_RoleMapping class. Deletes all contained elements.

char * GetName();

Returns a pointer to a copy of the name of the role associated to this mapping.

int SetName(**char** *);

Assigns a new name to the role associated with this role mapping. The method returns `OK` if the operation was successful, `ERROR` otherwise.

ModeFE_NameList* GetRelatedClasses();

Returns a pointer to a copy of the list of the labels which identify candidate Managed Object Classes in this role. If no class exists, the method returns an empty list.

int AddRelatedClass(ModeFE_Label *);

Adds one class label to the head of the list of candidate classes for this role. The method returns `OK` if the operation was successful, `ERROR` otherwise.

int AppendRelatedClass(ModeFE_Label *);

Adds one class label to the end of the list of candidate classes for this role. The method returns `OK` if the operation was successful, `ERROR` otherwise.

int RemoveRelatedClass(ModeFE_Label *);

Removes the first occurence of a class label from the list of managed object class labels associated with the role mapping. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplaceRelatedClass(ModeFE_Label *, ModeFE_Label*);

Replaces the first occurence of the first parameter in the list of managed object class labels associated with the role type with the label given as a second parameter. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

void ClearRelatedClassesList();

Removes all labels from the list of candidata managed object classes for the role.

ModeFE_NameList* GetQualified();

Returns a pointer to a copy of the list of labels which qualify the relationship. If no attribute exists, the method returns an empty list.

int AddQualified(ModeFE_Label *);

Adds one attribute label to the head of the list of attributes labels which qualify the relationship in this role. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int AppendQualified(ModeFE_Label *);

Adds one attribute label to the end of the list of attributes labels which qualify the relationship in this role. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int RemoveQualified(ModeFE_Label *);

Removes the first occurence of the attribute label given in the methods parameter from the list of attributes which qualify the relationship in this role. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int ReplaceQualified(ModeFE_Label *, ModeFE_Label*);

Replaces the first occurence of the attribute label given in the first parameter in the list of attributes which qualify the relationship in this role with the second parameter. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

void ClearQualified();

Removes the labels of all qualifying attributes associated with the relationship class in this role.

ModeFE_Representation* GetRepresentation();

Returns the representation of this role mapping.

int SetRepresentation(ModeFE_Representation*);

Assigns a new representation to the role mapping. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int IsEqual(ModeFE_RoleMapping*);

Returns `TRUE` if the role mapping given in parameter is equal to the current one. Returns `FALSE` otherwise. The equality test is made only on the role name in this version of the library.

ModeFE_RoleMapping* Dupplicate();

Returns a pointer to a copy of the current object. The invoker is then responsible for deleting this pointer and its content.

void Print(ofstream *);

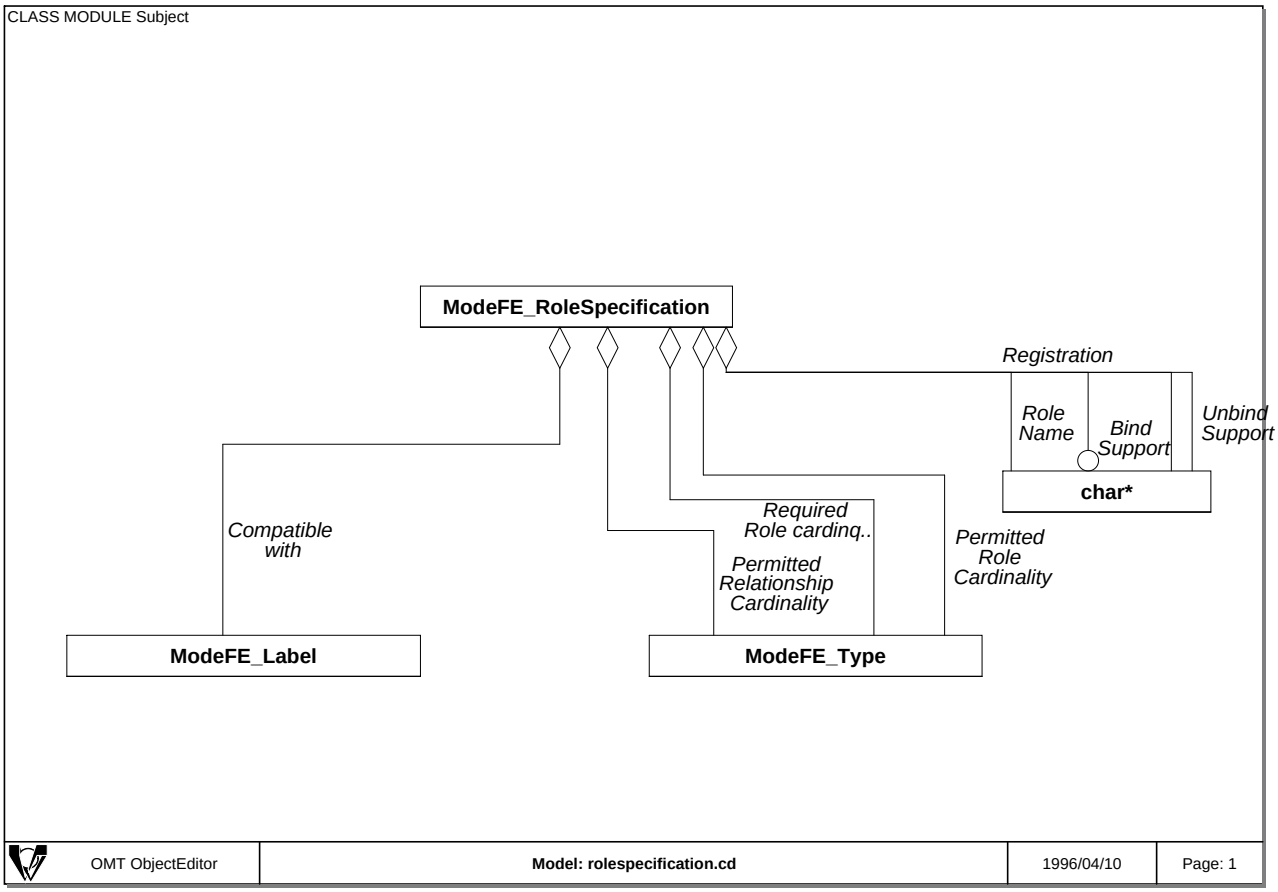
Prints in an ASCII format to the stream, the GRM specification that corresponds to this role mapping specification.

The ModeFE_RoleSpecification class

Purpose

The ModeFE_RoleSpecification stores information related to a role specification. This information covers the role name, the compatible class label if any, cardinality constraints, dinamical changes support information and the registration identifier if any.

Architecture



Available methods

```
ModeFE_RoleSpecification (char *,
                          ModeFE_Label *,
                          ModeFE_Type ,
                          ModeFE_Type ,
                          char *,
                          char *,
                          ModeFE_Type ,
                          char * );
```

Basic constructor of the ModeFE_RoleSpecification class. The parameters of the method qre:

1. the name of the role,
2. the label of the MOC that defines the compatibility to the role,
3. the permitted role cardinality,
4. the required role cadinality,
5. the support for binding, if any,
6. the support for dynamic unbinding, if any,
7. the permitted relationship cardinality
8. the registration identifier, if any (`NULL` if none).

```
ModeFE_RoleSpecification ();
```

Default constructor for the ModeFE_RoleSpecification class. Creates an empty role specification.

```
ModeFE_RoleSpecification (const ModeFE_RoleSpecification&);
```

Copy constructor of the ModeFE_RoleSpecification class. Returns no element in the current version of the library.

```
~ModeFE_RoleSpecification ();
```

Destructor of the ModeFE_RoleSpecification class. Deletes all contained elements.

```
char* GetName();
```

Returns a pointer to a copy of the name of the role.

```
int SetName(char*);
```

Assigns a new name to the role. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
ModeFE_Label* GetCompatibleWith();
```

Returns a pointer to a copy of the label describing the MOC that states the compatibility constraint for the role. If no MOC label is specified, the method returns an empty label.

```
int SetCompatibleWith(ModeFE_Label*);
```

Assigns a new label describing the MOC that states the compatibility constraint for the role. If no MOC label has to be specified, the parameter should be either `NULL` or an empty label. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
ModeFE_Type GetPermittedRoleCardinality();
```

Returns a pointer to a copy of the permitted role cadinality. If no permitted role cardinality is specified, the method returns `NULL`.

```
int SetPermittedRoleCardinality(ModeFE_Type);
```

Assigns a new value to the permitted role cardinality. If no value is expected, the parameter should be equal to `NULL`. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
ModeFE_Type GetRequiredRoleCardinality();
```

Returns a pointer to a copy of the required role cadinality. If no required role cardinality is specified, the method returns `NULL`.

```
int SetRequiredRoleCardinality(ModeFE_Type);
```

Assigns a new value to the required role cardinality. If no value is expected, the parameter should be equal to `NULL`. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
char* GetBindSupport();
```

Returns information on bind support in the role. If an operation is associated with the bind operation, the method returns a pointer to a copy of the operations name. If bind is supported but no operation is specified, the method returns a pointer to the `"NO_NAME_"` string. If Bind is not supported, the method returns `NULL`.

int SetBindSupport(**char***);

Assigns a new value to the bind support in the role. Possible values for the parameter are:

- If the bind operation has to be lonked to an operation, the parameter should contain the name of the operation,
- If bind has to be supported but no operation is specified, the value of the string should be `"NO_NAME_"`,
- If bind has not to be supported, the value of the parameter should be `NULL`.

This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

char* GetUnbindSupport();

Returns information on unbind support in the role. If an operation is associated with the unbind operation, the method returns a pointer to a copy of the operations name. If unbind is supported but no operation is specified, the method returns a pointer to the `"NO_NAME_"` string. If unbind is not supported, the method returns `NULL`.

int SetUnbindSupport(**char***);

Assigns a new value to the unbind support in the role. Possible values for the parameter are:

- If the unbind operation has to be lonked to an operation, the parameter should contain the name of the operation,
- If unbind has to be supported but no operation is specified, the value of the string should be `"NO_NAME_"`,
- If unbind has not to be supported, the value of the parameter should be `NULL`.

This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

ModeFE_Type GetPermittedRelationshipCardinality();

Returns a pointer to a copy of the permitted relatiuonship cadinality. If no permitted relationship cardinality is specified, the method returns `NULL`.

int SetPermittedrelationshipCardinality(ModeFE_Type);

Assigns a new value to the permitted relationship cardinality. If no value is expected, the parameter should be equal to `NULL`. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

char* GetRegistration();

Returns a pointer to a copy of the role specification type registration identifier. The identifier is provided as a string and does not contain the "{" and "}" brackets.

void SetRegistration(**char***);

Assigns a new registration identifier to the role specification. The registration identifier should not include the start and stop brackets ("{","}").

ModeFE_RoleSpecification* Dupplicate();

Returns a pointer to a copy of the current object. The invoker is then responsible for deleting this pointer and its content.

int IsEqual(ModeFE_RoleSpecification*);

Returns `TRUE` if the role specification given in parameter is equal to the current one. Returns `FALSE` otherwise. The equality test is made only on the role name in this version of the library.

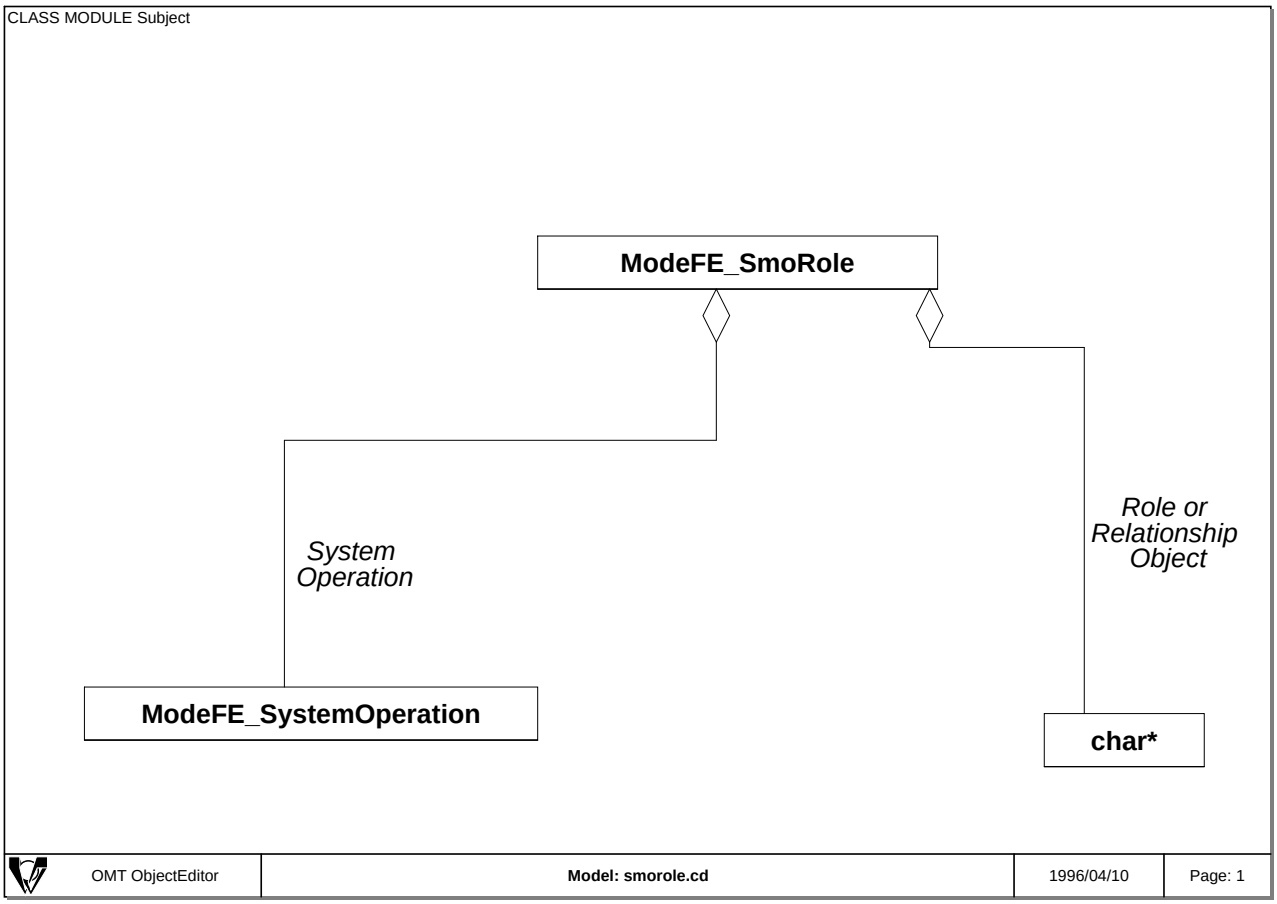
void Print (ofstream *) ;

Prints in an ASCII format to the stream, the GRM specification that corresponds to this role specification.

The ModeFE_SmoRole class

Purpose

The ModeFE_SmoRole class stores information on the association of a system operation and a role or a relationship object in the mapping part of a relationship operation mapping specification. The information provided in a ModeFE_SmoRole class is a system operation and the associated role or an indication if the association is a relationship object.



Available methods

ModeFE_SmoRole(ModeFE_SystemOperation*,
char *);

Basic constructor of the ModeFE_SmoRole class. The parameters are:

1. the system operation,

2. the information concerning the role or relationship object linked.

```
ModeFE_SmoRole();
```

Default constructor of the ModeFE_SmoRole class. Creates an empty instance.

```
ModeFE_SmoRole(const ModeFE_SmoRole&);
```

Copy constructor of the ModeFE_SmoRole class. Returns no element in the current version of the library.

```
~ModeFE_SmoRole();
```

Destructor of the ModeFE_SmoRole class. Deletes all contained elements.

```
ModeFE_SystemOperation* GetSysManOperation();
```

Returns a pointer to a copy of the system operation specification.

```
int SetSysManOperation(ModeFE_SystemOperation *);
```

Assigns a new system operation to the association. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
char* GetRoleOrRelationshipObject();
```

Returns either the name of the role linked to the system management operation or a statement on whether a relationship object is linked to the system management operation.
If a relationship object is concerned, the method returns the `NULL` value.

```
int SetRoleOrRelationshipObject(char*);
```

Assigns a new value to the system operation statement. The value for the parameter can be either `NULL` meaning that the relationship object is concerned or a string specifying the role name. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int IsEqual(ModeFE_SmoRole*);
```

Returns `TRUE` if the system management operation and the associated role or relationship class given in parameter is equal to the current one. Returns `FALSE` otherwise.

```
ModeFE_SmoRole* Dupplicate();
```

Returns a pointer to a copy of the current object. The invoker is then responsible for deleting this pointer and its content.

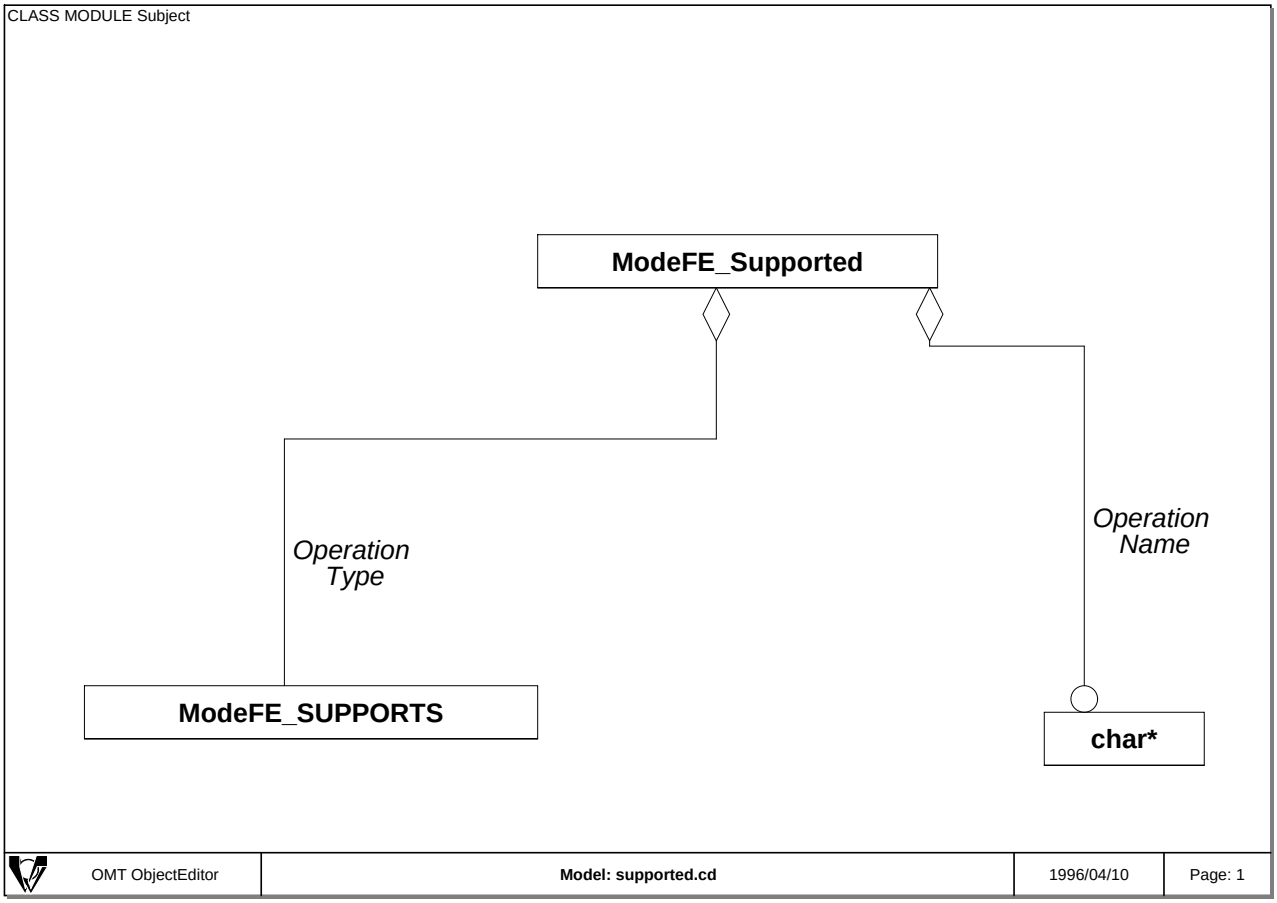
```
void Print ( ofstream * ) ;
```

Prints in an ASCII format to the stream, the GRM specification that corresponds to this system operation and role link type specification.

The ModeFE_Supported class

Purpose

The ModeFE_Supported class is used to store information concerning one relationship operation in the definition of a relationship class. A ModeFE_Supported instance stores a relationship operation type and an optional name.



Available methods

```
ModeFE_Supported ( ModeFE_SUPPORTS,
                   char *);
```

Basic constructor of the ModeFE_Supported class. The parameters of the method are:

1. the type of the relationship operation. Possible values are:
   - `ESTABLISH_`
   - `TERMINATE_`
   - `QUERY_`
   - `NOTIFY_`
   - `USER_DEFINED_`
2. the name associated with the operation (`NULL` if none.

```
ModeFE_Supported ();
```

Default constructor of the ModeFE_Supported class. Creates an empty instance.

```
ModeFE_Supported (const ModeFE_Supported&);
```

Copy constructor of the ModeFE_Supported class. Returns no element in the current version of the library.

~ModeFE_Supported ();

Destructor of the ModeFE_Supported class. Deletes all contained elements.

ModeFE_SUPPORTS GetSupportedOperation();

Returns the relationship operation type. Possible responses are:

- `ESTABLISH_`
- `TERMINATE_`
- `QUERY_`
- `NOTIFY_`
- `USER_DEFINED_`

int SetSupportedOperation(ModeFE_SUPPORTS);

Assigns a new type to the relationship instance. Possible values for the parameter are:

- `ESTABLISH_`
- `TERMINATE_`
- `QUERY_`
- `NOTIFY_`
- `USER_DEFINED_`

This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

char * GetOperationName();

Returns a pointer to a copy of the name associated with the operation type. If no name is associated, the method returns `NULL`

int SetOperationName(**char** *);

Assigns a new name to the operation type. If no name is expected the parameter must be equal to `NULL`. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

int IsEqual(ModeFE_Supported*);

Returns `TRUE` if the relationship operation given in parameter is equal to the current one. Returns `FALSE` otherwise. The equality test is made only on the relationship operation type in this version of the library.

ModeFE_Supported* Dupplicate();

Returns a pointer to a copy of the current object. The invoker is then responsible for deleting this pointer and its content.

void Print (ofstream *) ;

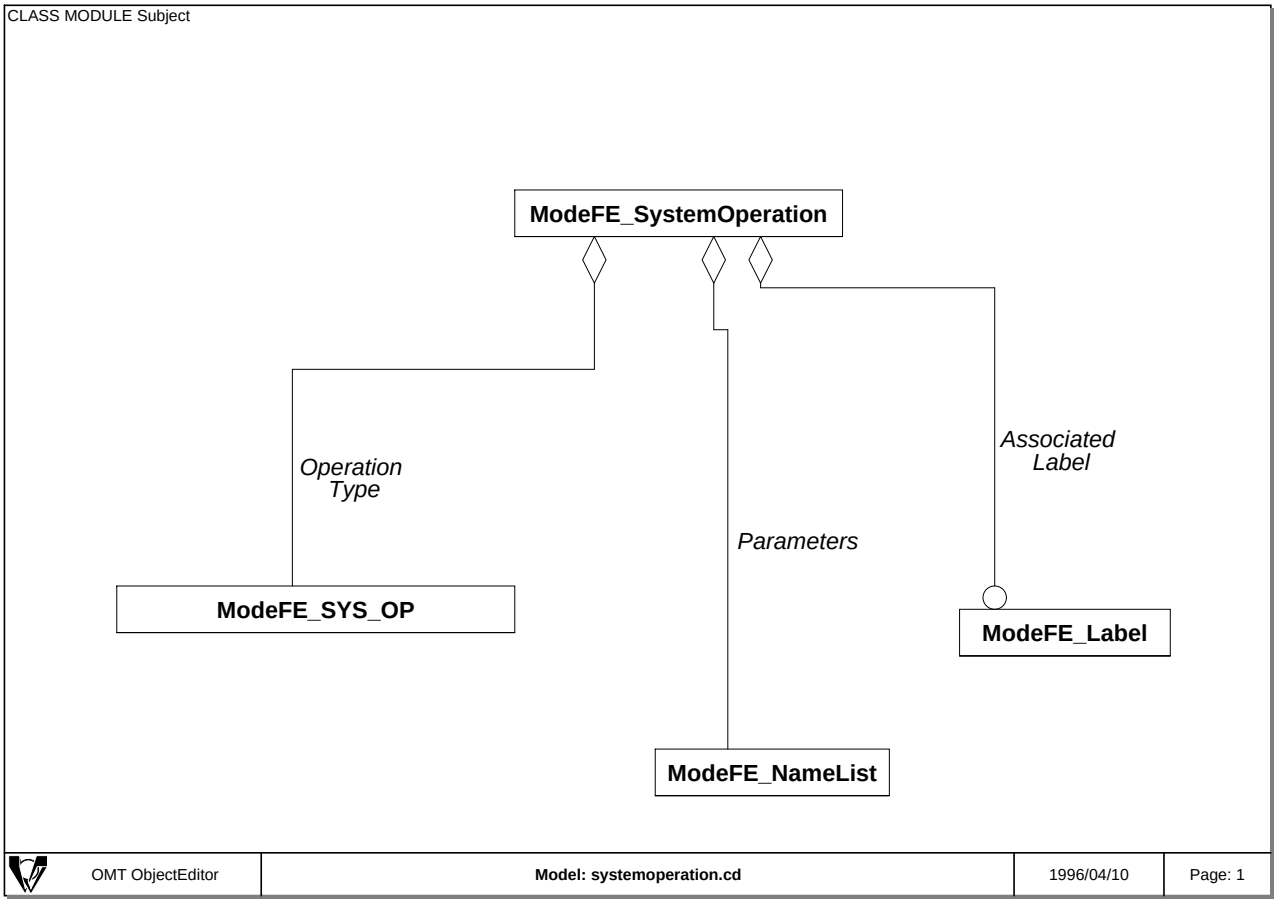
Prints in an ASCII format to the stream, the GRM specification that corresponds to this relationship operation type specification and the associated name in a relaitonship class specification.

The ModeFE_SystemOperation class

Purpose

The ModeFE_SystemOperation class stores information related to a system operation in the context of a relationship mapping specification. The values stored by an instance of this class are the system operation type and optional associated labels (attributes, actions and/or parameters).

Architecture



Available methods

```
ModeFE_SystemOperation(ModeFE_SYS_OP ,
                       ModeFE_Label *,
                       ModeFE_NameList *);
```

Basic constructor of the ModeFE_SystemOperation class. The parameters are:

1. the system operation type. Possible values are:
   - `OP_GET_`
   - `OP_REPLACE_`
   - `OP_ADD_`
   - `OP_REMOVE_`
   - `OP_CREATE_`
   - `OP_DELETE_`
   - `OP_ACTION_`

- OP_SET_
- OP_NOTIFICATION_

2. the label of either an action or an attribute associated with the operation (can be empty),
3. the list of parameters associated with this operation (can be empty).

ModeFE_SystemOperation();

Default constructor of the ModeFE_SystemOperation class. Creates an empty instance.

ModeFE_SystemOperation(**const** ModeFE_SystemOperation&);

Copy constructor of the ModeFE_SystemOperation class. Returns no element in the current version of the library.

~ModeFE_SystemOperation();

Destructor of the ModeFE_SystemOperation class. Deletes all contained elements.

ModeFE_SYS_OP GetOperationType();

Returns the operation type. Possible values for the response are:

- OP_GET_
- OP_REPLACE_
- OP_ADD_
- OP_REMOVE_
- OP_CREATE_
- OP_DELETE_
- OP_ACTION_
- OP_SET_
- OP_NOTIFICATION_

int SetOperationType(ModeFE_SYS_OP);

Assigns a new value to the operation type. Possible values for the parameter are:

- OP_GET_
- OP_REPLACE_
- OP_ADD_
- OP_REMOVE_
- OP_CREATE_
- OP_DELETE_
- OP_ACTION_
- OP_SET_

- `OP_NOTIFICATION_`

This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
ModeFE_Label* GetAttributeLabel();
```

Returns a pointer to a copy of the label of the element associated with the operation. This label can refer to:

- an attribute in the case of GET, REPLACE, ADD, REMOVE operation types,
- an action in the case of an ACTION operation type,
- a notification in the case of a NOTIFICATION operation type.
- a class label in the case of a CREATE operation type.

The label can be empty in the case of a CREATE operation type. In this case the method returns an empty label.

```
int SetAttributeLabel(ModeFE_Label *);
```

Assigns a new label associated with the operation type. The label can be empty or `NULL` only in the case of a CREATE operation type. This method returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
ModeFE_NameList* GetParameterList();
```

Returns a pointer to a copy of the list of parameter labels associated with the system operation. If no parameter is specified then the methods returns an empty list (see section on lists for details).

```
int AddParameter(ModeFE_Label*);
```

Adds one parameter label to the head of the list of parameters associated with the system operation. The methods returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int AppendParameter(ModeFE_Label*);
```

Adds one parameter label to the end of the list of parameters associated with the system operation. The methods returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int RemoveParameter(ModeFE_Label *);
```

Removes the first occurence of the parameter label given in parameter in the list of parameter labels associated with the system operation. The methods returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
int ReplaceParameter(ModeFE_Label*, ModeFE_Label *);
```

Replaces the first occurence of the parameter label given in the first parameter with the second parameter in the list of parameter labels associated with the system operation. The methods returns `OK` if the operation was successful. It returns `ERROR` otherwise.

```
void ClearParameters();
```

Removes all parameter labels from the list of parameters associated with the system operation.

```
ModeFE_SystemOperation* Dupplicate();
```

Returns a pointer to a copy of the current object. The invoker is then responsible for deleting this pointer and its content.

void Print (ofstream *) ;

Prints in an ASCII format to the stream, the GRM specification that corresponds to this system operation specification in the context of a relationship mapping specification.

A.2 Misc classes and functions

Within the library, one additional class and two functions are provided. The class is name `ModeFE_MessagePrinter` and is used within the parser to send the error message to the user. An instance of this class must be declared in the `main` program. This instance must be called `messagePrinter`.

A.3 Information Lists

Several list are used in the API. All the lists have a common interface. In this section we will describe a generic list interface as it is provided in the library.

A.3.1 The list architecture and available methods

Architecture

A list contains a head and an end. A list maintains a pointer to a location in the list which is incremented through each reading of the next element in the list. The methods are described in th next section.

Available methods

ModeFE_XXXList();

Default constructor for a list. Creates an empty list.

ModeFE_XXXList(XXX*);

Basic constructor for a list. Creates a list with the first element equal to the parameter.

ModeFE_XXXList(**const** ModeFE_XXXList&);

Copy constructor for a list. Returns no element in the current version of the library.

~ModeFE_XXXList();

Destructor of a list. Deletes all contained elements and the list support information.

int Add(ModeFE_XXX *);

Adds one element to the head of the list. The method returns `OK` if the operation was successful, `ERROR` otherwise.

int Append(ModeFE_XXX *);

Adds one element to the end of the list. The method returns `OK` if the operation was successful, `ERROR` otherwise.

int Remove(ModeFE_XXX *);

Removes the first occurence of the element given in parameter from the list. The method returns `OK` if the operation was successful, `ERROR` otherwise.

int Replace(ModeFE_XXX *, ModeFE_XXX *);

Removes the first occurence of the element given in the first parameter of the method with the element given as a second parameter in the list. The method returns `OK` if the operation was successful, `ERROR` otherwise.

int GoFirst();

Places the list pointer to the first element of the list.

ModeFE_XXX* GetElement();

Returns the current element in the list.

ModeFE_XXX* GetNextElement();

Returns the next element in the list. If there is no next element, the method returns the current one.

int IsEmpty();

Return `TRUE` if the list is empty, `FALSE` otherwise.

int IsLast();

Return `TRUE` if the current pointer points to the last element of the list, `FALSE` otherwise.

void Clear ();

Removes all elements from the list.

ModeFE_XXXList* Dupplicate();

Returns a pointer to a copy of the list. The invoker is responsible for deleting the returned pointer and its content.

int GetNumber();

Returns the number of elements in the list.

void Print(ofstream *);

Prints in an ASCII form the content of the list.

A.3.2 Available lists

In the table, all available lists in the library are presented in alphabetical order, the contained element type listed, an the purpose of the list given.

List	Element	Purpose
ModeFE_ActionList	ModeFE_Action	Stores a list of GDMO action specifications
ModeFE_AttributeAndPropertiesList	ModeFE_AttributeAndProperties	Stores attribute labels and the associated properties as they are specified in a GDMO package specification.
ModeFE_AttributeGroupList	ModeFE_AttributeGroup	Stores a list of GDMO Attribute Group specifications
ModeFE_AttributeList	ModeFE_Attribute	Stores a list of GDMO Attribute specifications
ModeFE_BehaviourList	ModeFE_Behaviour	Stores a list of GDMO Behaviour specifications
ModeFE_CondPackageList	ModeFE_CondPackage	Stores a list of Package labels and the associated condition as specified in a GDMO Managed Object Class specification.
ModeFE_FieldAttributeList	ModeFE_FieldAttributeList	Stores an attribute label and a information syntax field as specified in a notification template.
ModeFE_LabelAssociationList	ModeFE_LabelAssociationElement	Stores a list of labels and associated label list. Here a list of labels is associated to one label. This list is used in several templates. specifications
ModeFE_ManagedObjectClassList	ModeFE_ManagedObjectClass	Stores a list of GDMO Managed Object classes specifications
ModeFE_ModuleList	ModeFE_Module	Stores a list of loaded modules in a repository.
ModeFE_NameBindingList	ModeFE_NameBinding	Stores a list of Name-Binding Specifications.
ModeFE_NameList	ModeFE_Label	Stores a list of Labels. This list is used in many information classes.
ModeFE_NotificationList	ModeFE_Notification	Stores a list of GDMO Notification specifications.
ModeFE_OperationMappingList	ModeFE_OperationMapping	Stores a list of Operation mapping specifications as part of a relationship mapping specification.
ModeFE_PackageList	ModeFE_Package	Stores a list of GDMO package specifications.
ModeFE_ParameterList	ModeFE_Parameter	Stores a list of GDMO parameter specifications.
ModeFE_PropertyList	ModeFE_OneProperty	Stores a list of properties as they are associated with attributes in attribute and package specifications.
ModeFE_QualifierList	ModeFE_Qualifier	
ModeFE_RelationshipClassList	ModeFE_RelationshipClass	Stores a list of GRM Relationship class specifications.
ModeFE_RelationshipMappingList	ModeFE_RelationshipMapping	Stores a list of GRM Relationship mapping specifications.
ModeFE_RoleMappingList	ModeFE_RoleMapping	Stores a list of role mapping specifications as they are specified in relationship mappings.
ModeFE_RoleSpecificationList	ModeFE_RoleSpecification	Stores a list of role specifications as they are described in a relatioship specification.
ModeFE_SmoRoleList	ModeFE_SmoRole	
Stores a list of system management operations and the roles to which they apply in a relationship mapping specification.		
ModeFE_SupportedList	ModeFE_Supported	stores a list of relationship operation type and an optional name as they are specified in a relationship class.

Appendix B

A small example of the use of ModeFE: the basic pretty printer

Within this chapter we present an example on how to integrate and use the Library through a simple example in which we load a module and if the parsing was successful, print the specification formatted in TeX to a file. The code corresponds to the complete example given with the library.

B.1 The code

```
//
//          MODE-FE
//       Authors : Olivier  Festor
//                 Emmanuel Nataf
//
//       Copyright 1996 by Institut National de Recherche en Informatique
//                         et en Automatique (INRIA)
//       All rights reserved.  See COPYRIGHT in top-level directory.
//
// File:         ModeFE_Mainparser.cc
// Description: VT100 Parser Implementation File.
// Revision:     1.0
// Contact:      Olivier Festor
//               INRIA Lorraine
//               Technopole de Nancy-Brabois
//               - Campus scientifique -
//               615, Rue du Jardin Botanique B.P. 101
//               54600 Villers-Les-Nancy Cedex
//               FRANCE
//               tel: (+33) 83.59.20.16
//               fax: (+33) 83.27.83.19
//               e-mail: festor@loria.fr

#include <stdio.h>
#include <iostream.h>
#include <fstream.h>
#include "ModeFE_AllClasses.hh"
#include "ModeFE_Grammar.bison.tab.h"
#include "ModePP_TexPrinter.hh"
#include "ModeFE_MessagePrinter.hh"

extern char* strcopy(char*);
extern "C" int strcasecmp(const char *, const char * );
```

```
parse Parser;
ModeFE_Repository repository;
ModeFE_MessagePrinter messagePrinter;

int main(int argc, char **argv)
{
 FILE *fp;
 char* filename;
 char* outfile;
 ModePP_TexPrinter pout;
 int erreur;
 if (argc ≠ 4)
   {
     cout ≪ endl ≪ endl ≪ "Usage:  Main <grm-gdmo-file-name> -o <output-TeX-file>" ≪ endl ≪
endl;
     cout ≪ "Try again!" ≪ endl ≪ endl;
     exit(1);
   }
 else
   {
     filename = strcopy(argv[1]);
     if (strcasecmp(argv[2],"-o") ≠ 0)
       {
         cout ≪ endl ≪ "Usage:  Main <grm-gdmo-file-name> -o <output-TeX-file>" ≪ endl ≪
endl;
         cout ≪ "Try again!" ≪ endl ≪ endl;
         exit(1);
       }
     else
       {
         outfile = strcopy(argv[3]);
         if ((fp = freopen(filename,"r",stdin)) == NULL)
           return ERROR ;
         else
           {
             if ((erreur = Parser.yyparse()) == 0)
               {
                 ofstream OutFile ( outfile, ios::out );
                 cout ≪ "Parsing was Sucessful !!!" ≪ endl;
                 cout ≪ "...Nice job !!" ≪ endl;
                 cout ≪ "......You are an OSI Guru!!!!" ≪ endl;
                 ModeFE_Module* res = (repository.GetModules())→GetElement();
                 pout.PrintOSITexModule(&OutFile,res);
               }
             else
               {
                 cout ≪ "Parsing was unsuccessful !!!" ≪ endl;
                 cout ≪ "...You should look at you GRM/GDMO specification!!" ≪ endl;
               };
             fclose(fp);
           }
       }
   }
 cout ≪ "Thanks for having used MODE-FE !\n\n\n\n";
};
```

B.2 Comments

B.2.1 Required instances in any application

in any application using the ModeFE-Library following statements are required:

```
#include <stdio.h>
#include <iostream.h>
#include <fstream.h>
#include "ModeFE_AllClasses.hh"
#include "ModePP_TexPrinter.hh"
```

Following instances are required as global objects in the main application:

```
parse Parser;                                          // the parser
ModeFE_Repository repository;                          // the repository
ModeFE_MessagePrinter messagePrinter;                  // the message printer used by
the parser
```

Invocation to the parser is made through the following command:

```
Parser.yyparse()                                       // invocation to the parser
```

Unit´e de recherche INRIA Lorraine, Technopˆole de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhˆone-Alpes, 46 avenue F´elix Viallet, 38031 GRENOBLE Cedex 1
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399