HAL
open science

The GDMO and GRM Modules Semantic Checker of the MODERES Java Toolkit

Olivier Festor

To cite this version:

Olivier Festor. The GDMO and GRM Modules Semantic Checker of the MODERES Java Toolkit. [Technical Report] RT-0208, INRIA. 1997, pp.17. inria-00069963

HAL Id: inria-00069963

https://inria.hal.science/inria-00069963v1

Submitted on 19 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

The GDMO and GRM Modules Semantic Checker of the MODERES Java Toolkit

Olivier Festor

N ° 0208

11 Juillet 1997

THÈME 1

Rapport technique

ISSN 0249-0803

The GDMO and GRM Modules Semantic Checker of the MODERES Java Toolkit

Olivier Festor

Thème 1 — Réseaux et systèmes
Projet RESEDAS

Rapport technique n0208 — 11 Juillet 1997 — 17 pages

Abstract: MODERES Java is an environment for the development of management information models. In its first release, it provides tools for the manipulation of information models specified using GDMO and GRM.

This report presents one component of the environment, i.e. the GDMO/GRM modules semantic checker. The report contains a detailled description of performed verifications as well as a complete implementation and usage guide for the semantic checker as implemented within the environment.

Key-words: GDMO, GRM, Java, MODERES, TMN

(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

Analyseur sémantique de modules GRM et GDMO dans l'environnement MODERES Java

Résumé : MODERES Java est un environnement de développement de modèles de l'information de gestion. Cet environnement fournit, dans sa première version, un ensemble d'outils pour la manipulation de modèles de l'information spécifiés en GDMO/GRM.

Ce rapport présente l'un des composants de l'environnement à savoir l'analyseur sématique des spécifications GDMO et GRM. Le rapport comporte une description détaillée des vérifications entreprises ainsi qu'une présentation de l'implémentation et de l'utilisation de l'analyseur.

Mots-clé : GDMO, GRM, Java, MODERES, RGT

1 Introduction

MODERES Java [Festor 96][Festor 97] is toolkit developped in our group to manipulate GDMO (Guidelines for the Definition of Managed Objects) [ISO-10165.4 92] (and lattest ammendments) as well as GRM (General Relationship Model) [CCITT.X.725 95] specifications. The toolkit is developped in Java and contains several facilities for mainpulating management information models.

This report presents all semantic checks performed by the semantic checker integrated in the environment. It also presents the architectutre of this semantic checker and provides a complete user's guide for this facility.

The report is organized as follows. Section 2 gives a detailled description of what we understand as an error free information model specification. Section 3 details how the semantic checker is implemented in the MODERES framework and how it can be invoked by users through the repository package API. Section 4 provides a template per template description of all performed ckecks and associated error messages. Section 5 provides information on currently unperformed semantic checks in the toolkit. Section 6 provides a conculsion to this first release.

2 Definition of an error free GDMO/GRM specification in the context of the MODERES semantic checker

Information models for OSI and TMN-based management as specified using the GDMO and GRM notations. One information model contains a set of definitions. One definition can be:

- a managed-object class,
- a package,
- a name-binding,
- an attribute,
- an attribute group,
- an action,
- a notification,
- a parameter,
- a behaviour,
- a relationship class,
- a relationship mapping.

Within each template, references to other templates can be made, e.g. within a class definition one can refer to package definitions through the labels. These references may be within the same module or refer to a label within another module. In the context of the MODERES semantic checker two kind of statements must be fulfilled by refering labels to pass the check. These statements are:

- all labels in a specification must refer to existing templates, i.e. loaded within the MODERES repository,
- all labels must be associated with templates which comply to the statement in which they are referenced within another template, e.g. in a DERIVED FROM clause of a MOC, only managed object class templates can be referenced.

One label can be defined only once in a module and referenced as often as needed. If one refers to a template in another module, the label must be prceeded by the module identifier. These are the main checks performed at the label and reference level. Several others concerning consistency checks are verified on a template by template basis as defined in the next sections.

3 Implementation and user's guide

The GDMO/GRM semantic checker is coded within the `GDMO_Repository` class of the MODERES core package. For a detailled description of the architecture of the MODERES Java toolkit, see [Festor 97].

The semantic checker is accessible through the invocation of the `checkSemantics()` method which returns a list of objects of type `GDMO_SemanticError`. If the list is empty, no error was detected. Otherwise, each element contains an explanation of a detected error.

Objects of type `GDMO_SemanticError` have the following fields:

- an error String which details the error.
- a reference to the template in which the error was detected,
- a string which gives the type of the template in which the error was detected.
- a reference to the label which has caused the error, if any (null if none).

Each `GDMO_SemanticError` object has also a printing facility through the `print` method. There the errors are printed onto the stream provided in the invocation parameter of the method.

A small example of the use of the semantics checker is given below. This is an excerpt from the code provided in the `BasicMIFGDMOGRMParser.java` file in the examples directory. All relevant parts of the code, are in bold. On can note at the end, how semantic checking is invoked over the repository and that the result of this invocation is of type `DList` which is a list of errors. If this list is empty, no error was found. Otherwise, all contained errors are of type `GDMO_SemanticError` and the examples prints all errors to a file by iterating over the error list returned by the semantics checker.

```
import FR.loria.resedas.moderes.repository.*;
import FR.loria.resedas.moderes.frontend.*;
import java.io.*;
import COM.objectspace.jgl.*;

/** Basic multi file parser
 * Loads several GDMO/GRM files, parses the files and performs semantics
 * checks. Errors are written to the output file stated after the -o statement
 * of the command line.
 * @author Olivier Festor
 * @version 0.9
 */

public class BasicMIFGDMOGRMParser
{
  /** Main method of the parser
   * @param args list of arguments
   */
  public static void main(String args[])
  {
    // Declares the parser to be used
    MODERESTextGDMOGRMParser parser;

    // Instanciates a repository to be filled
    GDMO_Repository repository = new GDMO_Repository();

     // Creates the parser and opens input file
     try
     {
       parser = new MODERESTextGDMOGRMParser(new java.io.FileInputStream(args[0]));
       } catch (java.io.FileNotFoundException e)
     {
       System.out.println("GDMO/GRM Parser Version 0.9:  File " + args[0] + " not found.");
       return;
       }
     // Parsing starts here
```

```
    int i = 0;
    while ( (i<args.length) && (args[i].equals("-o") != true))
    {
      System.out.println(" Reading input specification from " + args[i]);
      System.out.println(" Start parsing ....... ");
      try // Creates the parser and opens input file
      {
        parser.ReInit(new java.io.FileInputStream(args[i]));

        } catch (java.io.FileNotFoundException e)
      {
        System.out.println("GDMO/GRM Parser Version 0.9:  File " + args[0] + " not found.");
        return;
        }
      try // Parsing
      {
        // creates the module which has to be filled
      GDMO_Module module = new GDMO_Module();
      parser.ParseGDMOGRMspecification(module);
      // adds the module to the repository if parsing was successfull
      repository.getModules().add(module);
        System.out.println("MODERES GDMO/GRM Parser Version 0.9:'')
        System.out.println( ``specification " + args[i] + " parsed successfully.");
        i++;
        } catch (ParseError e)
      {
        System.out.println("GDMO/GRM Parser Version 0.9:  exit due to parse error.");
        i++;
        }
      };
    System.out.println(" Start semantics checks over the repository ");
   DList result = repository.checkSemantics();
   if (result.isEmpty() != true)
   {
    System.out.println(" Semantic errors found .....");
    DListIterator iterator = result.begin();
    GDMO_SemanticError err = null;
    // iterating over all errors
    try {
      FileOutputStream ofile = new FileOutputStream(args[i+1]);
      PrintStream outstr = new PrintStream(ofile);
      while ( iterator.hasMoreElements() )
      {
       err = (GDMO_SemanticError) iterator.nextElement();
       err.print(outstr);
       }
      }
      catch (java.io.IOException e)
      {
        System.out.println("File not found and IO/Error in the module");
        return;
        };
      };
    };
  };
};
```

In addition of performing semantical checking, the semantic checker of the MODERES Java toolkit also extends the repository in solving all references, i.e. each time a label is referred within a template, the checker associates to the label a direct way of getting the template. Thus, when one uses the repository to access specifications, from a label, one can access the associated template through the `getReference()` method. If

semantic checking was successfull, all references are solved. Otherwise only those which have not raised an error are solved.

4 Performed semantic checks

In GDMO as well as in GRM, all templates do not require any semantic checks. In the first release of the GDMO/GRM semantic checker, we perform several verifications but not all possible verifications due to the abscence in the first release of the environment of an ASN.1 parser. In this section, we detail for each template, which semantic checks are preformed.

4.1 The Action template

Concerning action templates, following verifications are made by the semantic checker for each template specification:

- all behaviour definitions in the `BEHAVIOUR` clause must refer to behaviour templates and these templates must exist.

  if an error is detected, the error object contains following values:

  - message: `"Unresolved Behaviour Reference!"`,
  - template reference: the action template in which the error was detected,
  - template type: `"ACTION"`,
  - error label: a reference to the label which was not resolved.

- all parameter[1] definitions in the `PARAMETERS` clause must refer to parameter templates and these templates must exist.

  if an error is detected, the error object exhibits following values:

  - message: `"Unresolved Parameter Reference!"`,
  - template reference: the action template in which the error was detected,
  - template type: `"ACTION"`,
  - error label: a reference to the label which was not resolved.

- if a reply syntax is defined, then the `CONFIRMED` mode must be set in the action definition.

  if an error is detected, the error object exhibits following values:

  - message: `"Reply-Syntax/Invocation Mode Conflict!"`,
  - template reference: the action template in which the error was detected,
  - template type: `"ACTION"`,
  - error label: `null`.

4.2 The Attribute template

Concerning attribute templates, following verifications are made by the semantic checker for each template specification:

- all behaviour definitions in the `BEHAVIOUR` clause must refer to behaviour templates and these templates must exist.

  if an error is detected, the error object exhibits following values:

  - message: `"Unresolved Behaviour Reference!"`,
  - template reference: the attribute template in which the error was detected,
  - template type: `"ATTRIBUTE"`,

[1] Additional checks must be performed over parameters in this clause. See Additional checks section for more details.

  - error label: a reference to the label which was not resolved.

- all parameter definitions[2] in the `PARAMETERS` clause must refer to parameter templates and these templates must exist.

  if an error is detected, the error object exhibits following values:

  - message: `"Unresolved Parameter Reference!"`,
  - template reference: the attribute template in which the error was detected,
  - template type: `"ATTRIBUTE"`,
  - error label: a reference to the label which was not resolved.

4.3 The Attribute Group template

Only one semantic check is performed over attribute group templates. This check ensures that all attributes referenced within the attribute group are real attribute templates and these templates exist.

If an error is detected, the error object exhibits following values:

- message: `"Unresolved Attribute Reference!"`,
- template reference: the attribute group template in which the error was detected,
- template type: `"ATTRIBUTE-GROUP"`,
- error label: a reference to the label which was not resolved.

4.4 The Managed Object Class template

Concerning Managed Object Class templates, following verifications are made by the semantic checker for each template specification:

- all labels in the `DERIVED FROM` CLAUSE must refer to Managed Object templates and these templates must be loaded in the system.

  if an error is detected, the error object exhibits following values:

  - message: `"Unresolved Managed Object Class Reference!"`,
  - template reference: the MO in which the error was detected,
  - template type: `"MANAGED-OBJECT-CLASS"`,
  - error label: a reference to the label which was not resolved.

- both references to mandatory and conditional packages must refer to package definitions and these packages must be loaded within the system[3].

  if an error is detected, the error object exhibits following values:

  - message: `"Unresolved Package Reference!"` or `"Unresolved Conditional Package Reference!"` in case of a conditional package,
  - template reference: the MO in which the error was detected,
  - template type: `"MANAGED-OBJECT-CLASS"`,
  - error label: a reference to the label which was not resolved.

- all behaviour definitions in the `BEHAVIOUR` clause must refer to behaviour templates and these templates must exist.

  if an error is detected, the error object exhibits following values:

  - message: `"Unresolved Behaviour Reference!"`,
  - template reference: the MO in which the error was detected,
  - template type: `"MANAGED-OBJECT-CLASS"`,
  - error label: a reference to the label which was not resolved.

[2] Like for actions, additional verifications have to be made on the content of the parameters associated to atributes. These checks are presented in section 5.

[3] Additional checks must be performed on packages within managed object classes. See 5 section for details.

4.5 The Name-Binding template

Following verifications are performed over name-binding templates:

- the label referenced in the `SUBORDINATE OBJECT CLASS` template must refer to a Managed Object Class template and this template must exist.

  if an error is detected, the error object exhibits following values:

  - message: `"Unresolved MOC Reference!"`,
  - template reference: the name-binding template in which the error was detected,
  - template type: `"NAME-BINDING"`,
  - error label: a reference to the label which was not resolved.

- the label referenced in the `NAMED BY SUPERIOR OBJECT CLASS` template must refer to a Managed Object Class template and this template must exist.

  if an error is detected, the error object exhibits following values:

  - message: `"Unresolved MOC Reference!"`,
  - template reference: the name-binding template in which the error was detected,
  - template type: `"NAME-BINDING"`,
  - error label: a reference to the label which was not resolved.

- all labels in the `BEHAVIOUR` part must refer to behaviour templates and these templates must exist.

  if an error is detected, the error object exhibits following values:

  - message: `"Unresolved Behaviour Reference!"`,
  - template reference: the name-binding template in which the error was detected,
  - template type: `"NAME-BINDING"`,
  - error label: a reference to the label which was not resolved.

- the label in the `WITH ATTRIBUTE` clause must refer to an attribute template and this template must exist within the repository[4].

  if an error is detected, the error object exhibits following values:

  - message: `"Unresolved Attribute Reference!"`,
  - template reference: the name-binding template in which the error was detected,
  - template type: `"NAME-BINDING"`,
  - error label: a reference to the label which was not resolved.

- all labels in the `CREATE` clause must refer to existing parameters in the repository[5].

  if an error is detected, the error object exhibits following values:

  - message: `"Unresolved Parameter Reference!"`,
  - template reference: the name-binding template in which the error was detected,
  - template type: `"NAME-BINDING"`,
  - error label: a reference to the label which was not resolved.

- all labels in the `DELETE` clause must refer to existing parameters in the repository[6].

  if an error is detected, the error object exhibits following values:

  - message: `"Unresolved Parameter Reference!"`,
  - template reference: the name-binding template in which the error was detected,
  - template type: `"NAME-BINDING"`,
  - error label: a reference to the label which was not resolved.

[4] There are additional checks that must be performed on this attribute and the associated subordinate object class. These checks are not performed in the first release of the semantic checker. See the 5 section of this report for a detailled description of these additional verifications

[5] Several additional checks must be performed over parameters in this clause. See section 5 for details.

[6] Several additional checks must be performed over parameters in this clause. See section 5 for details.

4.6 The Notification template

Concerning notification templates, following verifications are made by the semantic checker for each template specification:

- all behaviour definitions in the `BEHAVIOUR` clause must refer to behaviour templates and these templates must exist.

  if an error is detected, the error object exhibits following values:

  - message: `"Unresolved Behaviour Reference!"`,
  - template reference: the notification template in which the error was detected,
  - template type: `"NOTIFICATION"`,
  - error label: a reference to the label which was not resolved.

- all parameter definitions in the `PARAMETERS` clause must refer to parameter templates and these templates must exist[7].

  if an error is detected, the error object exhibits following values:

  - message: `"Unresolved Parameter Reference!"`,
  - template reference: the notification template in which the error was detected,
  - template type: `"NOTIFICATION"`,
  - error label: a reference to the label which was not resolved.

- all attribute labels in the `ATTRIBUTE IDS` must refer to GDMO attributes and these attributes must exist.

  if an error is detected, the error object exhibits following values:

  - message: `"Unresolved Attribute Reference!"`,
  - template reference: the notification template in which the error was detected,
  - template type: `"NOTIFICATION"`,
  - error label: a reference to the label which was not resolved.

4.7 The Package template

Following checks are performed within a package template definition:

- all behaviour labels in in the behaviour part must refer to existing behaviour definitions

  if an error is detected, the error object exhibits following values:

  - message: `"Unresolved Behaviour Reference!"`,
  - template reference: the package template in which the error was detected,
  - template type: `"PACKAGE"`,
  - error label: a reference to the label which was not resolved.

- all attributes labels specified in the attributes clause refer to existing attribute templates.

  if an error is detected, the error object exhibits following values:

  - message: `"Unresolved Attribute Reference!"`,
  - template reference: the package template in which the error was detected,
  - template type: `"PACKAGE"`,
  - error label: a reference to the label which was not resolved.

[7] Several additional checks must be performed over parameters in this clause. See section 5 for details.

- all parameters labels associated to an attribute in the attributes clause are real parameters and these parameters templates must exist[8].

  if an error is detected, the error object exhibits following values:

  - message: **`"Unresolved Parameter Reference!"`**,
  - template reference: the package template in which the error was detected,
  - template type: **`"PACKAGE"`**,
  - error label: a reference to the label which was not resolved.

- if a derivation rule is specified within a value reference of an attribute and if this rule refers to a behaviour template, the template must be of type behaviour and must be provided in the repository.

  if an error is detected, the error object exhibits following values:

  - message: **`"Unresolved Behaviour Reference!"`**,
  - template reference: the package template in which the error was detected,
  - template type: **`"PACKAGE"`**,
  - error label: a reference to the label which was not resolved.

- all attribute groups labels defined in the attribute group clause must refer to existing attribute groups and these attribute groups must be provided in the repository.

  if an error is detected, the error object exhibits following values:

  - message: **`"Unresolved Attribute Group Reference!"`**,
  - template reference: the package template in which the error was detected,
  - template type: **`"PACKAGE"`**,
  - error label: a reference to the label which was not resolved.

- attribute groups can only be extended with additional attributes if their specification does not exhibit the FIXED statement.

  if an error is detected, the error object exhibits following values:

  - message: **`"Unextensible Attribute Group!"`**,
  - template reference: the package template in which the error was detected,
  - template type: **`"PACKAGE"`**,
  - error label: a reference to the label which was not resolved.

- all attributes which extend attribute groups are real attribute definitions and these definitions must exist.

  if an error is detected, the error object exhibits following values:

  - message: **`"Unresolved Attribute Reference!"`**,
  - template reference: the package template in which the error was detected,
  - template type: **`"PACKAGE"`**,
  - error label: a reference to the label which was not resolved.

- all actions defined in the actions clause are real action templates and these templates must exist within the repository.

  if an error is detected, the error object exhibits following values:

  - message: **`"Unresolved Action Reference!"`**,
  - template reference: the package template in which the error was detected,
  - template type: **`"PACKAGE"`**,
  - error label: a reference to the label which was not resolved.

[8] Several additional checks must be performed over parameters in this clause. See section 5 for details.

- all parameters which extend actions must exist[9].

  if an error is detected, the error object exhibits following values:

  - message: **`"Unresolved Parameter Reference!"`**,
  - template reference: the package template in which the error was detected,
  - template type: **`"PACKAGE"`**,
  - error label: a reference to the label which was not resolved.

- all notifications referenced in the notifications clause must be real notifications and these notifications must exist.

  if an error is detected, the error object exhibits following values:

  - message: **`"Unresolved Notification Reference!"`**,
  - template reference: the package template in which the error was detected,
  - template type: **`"PACKAGE"`**,
  - error label: a reference to the label which was not resolved.

- all parameter which are associated to a notification in the package, must refer to real parameter templates and these templates must exist[10].

  if an error is detected, the error object exhibits following values:

  - message: **`"Unresolved Parameter Reference!"`**,
  - template reference: the package template in which the error was detected,
  - template type: **`"PACKAGE"`**,
  - error label: a reference to the label which was not resolved.

4.8 The Parameter template

Following verifications are undertaken by the semantics checker:

- all behaviour labels in the `BEHAVIOUR` clause must be associated to real behaviour templates.

  if an error is detected, the error object exhibits following values:

  - message: **`"Unresolved Behaviour Reference!"`**,
  - template reference: the parameter template in which the error was detected,
  - template type: **`"PARAMETER"`**,
  - error label: a reference to the label which was not resolved.

- if the parameter is defined from an attribute, then the label of the attribute clause must refer to an attribute definition and this definition must exist.

  if an error is detected, the error object exhibits following values:

  - message: **`"Unresolved Attribute Reference!"`**,
  - template reference: the parameter template in which the error was detected,
  - template type: **`"PARAMETER"`**,
  - error label: a reference to the label which was not resolved.

[9] Several additional checks must be performed over parameters in this clause. See section 5 for details.
[10] Several additional checks must be performed over parameters in this clause. See section 5 for details.

4.9 The relationship class template

Following verifications are performed by the semantic checker[11]:

- all labels in the `DERIVED FROM` clause must refer to existing relationship class templates.

  if an error is detected, the error object exhibits following values:

  - message: `"Unresolved Relationship Class Reference!"`,
  - template reference: the relationship class template in which the error was detected,
  - template type: `"RELATIONSHIP CLASS"`,
  - error label: a reference to the label which was not resolved.

- all labels in the `BEHAVIOUR` clause must refer to existing behaviour templates

  if an error is detected, the error object exhibits following values:

  - message: `"Unresolved behaviour Reference!"`,
  - template reference: the relationship class template in which the error was detected,
  - template type: `"RELATIONSHIP CLASS"`,
  - error label: a reference to the label which was not resolved.

- all labels in the `QUALIFIER` clause must refer to existing attributes.

  if an error is detected, the error object exhibits following values:

  - message: `"Unresolved Attribute Class Reference!"`,
  - template reference: the relationship class template in which the error was detected,
  - template type: `"RELATIONSHIP CLASS"`,
  - error label: a reference to the label which was not resolved.

- roles:

  following checks are performed upon role specifications:

  - if the `COMPATIBLE WITH` clause is present, then the class label must refer to an existing MOC template[12].

    if an error is detected, the error object exhibits following values:

    * message: `"Unresolved Managed Object Class Reference!"`,
    * template reference: the relationship class template in which the error was detected,
    * template type: `"RELATIONSHIP CLASS"`,
    * error label: a reference to the label which was not resolved.

4.10 The relationship mapping template

Following verifications are performed by the checker over relationship mapping templates:

- the label in the `RELATIONSHIP CLASS` clause must refer an existing relationship class template.

  if an error is detected, the error object exhibits following values:

  - message: `"Unresolved Relationship Class Reference!"`,
  - template reference: the relationship mapping template in which the error was detected,
  - template type: `"RELATIONSHIP-MAPPING"`,
  - error label: a reference to the label which was not resolved.

- all labels in the `BEHAVIOUR` clause must refer to existing behaviour templates

  if an error is detected, the error object exhibits following values:

[11] Additional checks should be performed over relationship class templates. See 5 section for details.
[12] Additional checks should be performed over relationship class templates. See 5 section for details.

  - message: **"Unresolved Behaviour Reference!"**,
  - template reference: the relationship mapping template in which the error was detected,
  - template type: **"RELATIONSHIP-MAPPING"**,
  - error label: a reference to the label which was not resolved.

- the label of the `RELATIONSHIP OBJECT` must be a managed object class template[13].

  if an error is detected, the error object exhibits following values:

  - message: **"Unresolved Managed Object Class Reference!"**,
  - template reference: the relationship mapping template in which the error was detected,
  - template type: **"RELATIONSHIP-MAPPING"**,
  - error label: a reference to the label which was not resolved.

- qualifiers in the `QUALIFIES` part of a `RELATIONSHIP OBJECT` clause must refer to existing attribute templates[14].

  if an error is detected, the error object exhibits following values:

  - message: **"Unresolved Attribute Reference!"**,
  - template reference: the relationship mapping template in which the error was detected,
  - template type: **"RELATIONSHIP-MAPPING"**,
  - error label: a reference to the label which was not resolved.

- each role name in a role mapping must refer to an existing role name in the associated relationship class.

  if an error is detected, the error object exhibits following values:

  - message: **"Undefined Role"** + roleName,
  - template reference: the relationship mapping template in which the error was detected,
  - template type: **"RELATIONSHIP-MAPPING"**,
  - error label: null

- all labels in the `RELATED-CLASSES` of a role mapping specification must refer to managed object class templates[15].

  if an error is detected, the error object exhibits following values:

  - message: **"Undefined Managed Object Class Reference!"**,
  - template reference: the relationship mapping template in which the error was detected,
  - template type: **"RELATIONSHIP-MAPPING"**,
  - error label: a reference to the label which was not resolved.

- if the representation of the role mapping is of type NAMING, the associated label must refer to a Name-Binding template

  if an error is detected, the error object exhibits following values:

  - message: **"Undefined Name-Binding Reference!"**,
  - template reference: the relationship mapping template in which the error was detected,
  - template type: **"RELATIONSHIP-MAPPING"**,
  - error label: a reference to the label which was not resolved.

- if the representation of the role mapping is of type ATTRIBUTE or RELATIONSHIP-OBJECT-USING-POINTER, the associated label must refer to an attribute template.

  if an error is detected, the error object exhibits following values:

[13] Additional checks should be performed over the relationship object, if present. See 5 section for details.
[14] Additional checks should be performed over those attributes. See 5 section for details.
[15] Additional checks should be performed over those managed object class templates. See 5 section for details.

  - message: **“Undefined Attribute Reference!”**,
  - template reference: the relationship mapping template in which the error was detected,
  - template type: **“RELATIONSHIP-MAPPING”**,
  - error label: a reference to the label which was not resolved.

- all qualifying labels in the **QUALIFIES** clause of a role mapping must refer to attribute templates[16].

  if an error is detected, the error object exhibits following values:

  - message: **“Undefined Attribute Reference!”**,
  - template reference: the relationship mapping template in which the error was detected,
  - template type: **“RELATIONSHIP-MAPPING”**,
  - error label: a reference to the label which was not resolved.

- all attribute in the system-management operation part of a mapping must refer to existing specifications of the attribute type[17].

  if an error is detected, the error object exhibits following values:

  - message: **“Undefined Attribute Reference!”**,
  - template reference: the relationship mapping template in which the error was detected,
  - template type: **“RELATIONSHIP-MAPPING”**,
  - error label: a reference to the label which was not resolved.

- all parameters in the system-management operation part of a mapping must refer to existing specifications of the parameter type[18].

  if an error is detected, the error object exhibits following values:

  - message: **“Undefined Parameter Reference!”**,
  - template reference: the relationship mapping template in which the error was detected,
  - template type: **“RELATIONSHIP-MAPPING”**,
  - error label: a reference to the label which was not resolved.

- all role names in the **maps-to** part of an operation mapping must exist in the associated relationship class.

  if an error is detected, the error object exhibits following values:

  - message: **“Undefined Role !”** + *roleName*,
  - template reference: the relationship mapping template in which the error was detected,
  - template type: **“RELATIONSHIP-MAPPING”**,
  - error label: null

5 Additional semantic checks

In the first release of the semantic checker several verifications are not performed. Those checks are:

- for **action templates:** all parameters referred in the parameters clause must comply in their context with the field in which they occur in the action.

  Following values are possible for the context:

  - ACTION-INFO,
  - ACTION-REPLY,
  - SPECIFIC-ERROR,

[16] Additional checks should be performed over those managed object class templates. See 5 section for details.
[17] Additional checks should be performed over those attribute templates. See 5 section for details.
[18] Additional checks should be performed over those parameter templates. See 5 section for details.

  - type-reference.

  this check must be further investigated. Especially it may be linked to the presence or abscence of an information or reply syntax in an action definition but this is unclearly defined in the GDMO standard.

- for **attribute templates:** all parameters referred in the parameters clause must comply in their context with attribute constraint. Thus those parameters can only exhibit the SPECIFIC-ERROR value in their context.

- for **managed object class templates:**

  - one can not derive twice from the same class at one inheritance level, i.e. one same class can not appear twice in the DERIVED FROM clause of a MOC.
  - a package defined as mandatory in one of the super-classes of the MOC, can not be defined as optional in the MOC template[19]

- for **name-binding templates:**

  - the attribute label defined in the `WITH ATTRIBUTE` clause must exist with the subordinate object class and referenced within one of its mandatory packages. This package may be defined either a in the managed object class itself or in one of its super-classes.
  - parameters associated to the CREATE clause of the name-binding template may only have a context exhibiting the SPECIFIC-ERROR value.
  - parameters associated to the DELETE clause of the name-binding template may only have a context exhibiting the SPECIFIC-ERROR value.

- for **notification templates:** all parameters referred in the parameters clause must comply in their context with the field in which they occur in the notification.

  Following values are possible for the context:

  - EVENT-INFO,
  - EVENT-REPLY,
  - SPECIFIC-ERROR,
  - type-reference.

  this check must be further investigated. Especially it may be linked to the presence or abscence of an information or reply syntax in a notification definition but this is unclearly defined in the GDMO standard. Moreover checks have probably to be performed over field names. This is left for further investigation.

- for **package templates:**

  - all parameters associated to an attribute within a package template may only have a context exhibiting the SPECIFIC-ERROR value.
  - all parameters associated to an action within a package template may only have a context exhibiting the following values:
    * ACTION-INFO,
    * ACTION-REPLY,
    * SPECIFIC-ERROR,
    * type-reference.

    this check must be further investigated. Especially it may be linked to the presence or abscence of an information or reply syntax in an action definition but this is unclearly defined in the GDMO standard.
  - all parameters associated to a notification within a package template may only have a context exhibiting the following values:

[19] Even if the inheritance rules defined within the OSI object model would solve this by forcing the package to be mandatory, we think its a conception error to define it as conditional in a subclass, and thus a semantic error has to be raised.

    * EVENT-INFO,
    * EVENT-REPLY,
    * SPECIFIC-ERROR,
    * type-reference.

    this check must be further investigated. Especially it may be linked to the presence or abscence of an information or reply syntax in a notification definition but this is unclearly defined in the GDMO standard.

- for **relationship class templates:**
  - if the class is derived from a super-class (DERIVED-FROM clause present) then no additional relationship notification (NOTIFY clause) can be specified in the supports clause. I.e. one can not add notifications to relationship subclasses (clause A.1.3.1-a of the GRM).
  - the MOC in the COMPATIBLE-WITH clause of a role must be compatible with those specified in the same clause and the same role in super-classes (clause A.1.3.1-d.2 of the GRM). Compatible means here either a subclass or an allomorphic class to those given in the same place in a super-class.
- for **relationship mapping templates:**
  - the MOC referenced in the RELATIONSHIP-OBJECT of a mapping shall be a subclass of `genericRelationshipObje` (Clause A.2.3.3 of the GRM).
  - the MOC referenced in the RELATIONSHIP-OBJECT of a mapping shall exhibit participant pointer attributes for each of the roles specified in the associated relationship class (Clause A.2.3.3 of the GRM).
  - attributes provided in extension of the RELATIONSHIP-OBJECT clause in a mapping must be labels defined in the QUALIFIED BY clause of the associated relationship class (Clause A.2.3.3 of the GRM).
  - MOC's in the RELATED-CLASSES of a role must be compatible with the MOC referenced in the COMPATIBLE WITH clause of the role in the associated relationship class (Clause A.2.3.4 of the GRM).
  - attributes provided in extension of the QUALIFIES construct clause in a role mapping must be labels defined in the QUALIFIED BY clause of the associated relationship class (Clause A.2.3.4 of the GRM).
  - each operation in the operation mapping part must be an operation defined in the associated relationship class specification.
  - **System Management operation:**
    * checks have to be performed on the existence of these attributes in associated managed object classes in the roles. This is left for further investigation.
    * parameters associated with system management operations must comply following constraints:
      · if a parameter is associated to the `GET`, `REPLACE`, `ADD` or `REMOVE` system operation, then its context may only exhibit the `SPECIFIC-ERROR` value.
      · if a parameter is associated to the `CREATE` system operation, then its context may exhibit only the `SPECIFIC-ERROR` value.
      · if a parameter is associated to the `DELETE` system operation, then its context may exhibit only the `SPECIFIC-ERROR` value.
      · if a parameter is associated to the `ACTION` system operation, then its context may exhibit only following values:
        `ACTION-INFO`, `ACTION-REPLY`, `SPECIFIC-ERROR`, `type-reference`.
      · if a parameter is associated to the `NOTIFICATION` system operation, then its context may exhibit only following values:
        `EVENT-INFO`, `EVENT-REPLY`, `SPECIFIC-ERROR` or `type-reference`.

Additional checks over permitted and required values in both GDMO and GRM should be performed but due to the abscence of an ASN.1 parser in this release of MODERES, those checks can not be made.

All these verifications, and probably new ones, will be provided in the next release of the checker.

6 Conclusion

In this report, we have presented the semantic checker implemented within the MODERES Java toolkit. This checker is implemented within the repository class of the moderes package. It performs many verifications over several modules and provides users with valuable information on semantical inconsistencies in their information model. As stated in the report, all existing semantic checks are not yet implement in the checker, although most are. These additional checks will be present in the nex release of the tool.

References

[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*, International Standard, CCITT.X.725, November 1995.

[Festor 96] O. Festor, E. Nataf et L. Andrey. MODE-FE: A GRM/GDMO Parser and its API -Release 1.0-
Reference Manual. Technical Report no. 0190, INRIA Lorraine, 1996.

[Festor 97] O. Festor. MODERES Java: Architecture and Core Packages. Rapport no. RT-0205, INRIA, May 1997.

[ISO-10165.1 92] International Organization for Standardization (ISO), *Structure of Management Information - Part 1: Management Information Model*, International Standard, ISO-10165.1, January 1992.

[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.

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, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
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