FCO-IM and UML

Investigation on the transformation of an elementary information grammar towards a class diagram

Elton Manoku
HAN University
Arnhem, 2002

1 Introduction

The aim of this material is to put a starting step in building a bridge between FCO-IM Diagrams and Object Oriented Modeling Diagrams. FCO-IM is powerful method to perform data analysis in conceptual layer. For displaying the designed data model in Entity Relationship Diagrams or ERD is already an algorithm. Even though many semantics are lost during this transformation, again displaying the result of analysis in ERD is wise because it is better known between data modelers and database designers around. Several tools are present where this algorithm is implemented.
Unified Programming Language or UML is gaining popularity nowadays and is becoming a standard in Object Oriented Modeling. UML is a composition of different diagrams: Activity Diagrams, Class Diagrams, Sequence Diagrams, State Diagrams, Use Case Diagrams.
These diagrams are used for different purposes. Mainly UML diagrams are used for software design and different kind of diagrams cover design issues in software development. For displaying data models in UML are used Class Diagrams. For this reason only this kind of diagram will be treated in this report. There are different tools, which are supporting UML.
One tool which make it possible to display class diagrams and after that converting those class diagrams in different platforms like Java, C++, XML-DTD, etc is Sybase Power Designer 9.
In this material, this tool has been seen as a proper one for displaying Class Diagrams converted from FCO-IM diagrams, because it seems to fully support UML syntax and gives the opportunity to access the repository of this tool through scripts or directly.
It will be treated in this material step by step an idea about the transformation FCO-IM Diagrams towards Class Diagrams.
In each step will be used student case example, extended with some more facts to include every possible situation like subtype-super type relationships.
Because the Class Diagram is not seen from a data treatment perspective, UML notation is extended with additional symbols to present better the Model in FCO-IM Diagram.
Because FCO-IM Case Tool that supports FCO-IM method has a generic repository and it is possible to access it, all needed queries to support this transformation will be given.
It will be explained the implementation of a tool supporting the transformation described below.

2 UML & Class Diagrams

UML is recently being used as a standard in software development to represent analysis design and implementation design.
UML is a composition of different diagrams:

- Activity Diagrams
- Class Diagrams
- Sequence Diagrams
- State Diagrams
- Use Case Diagrams

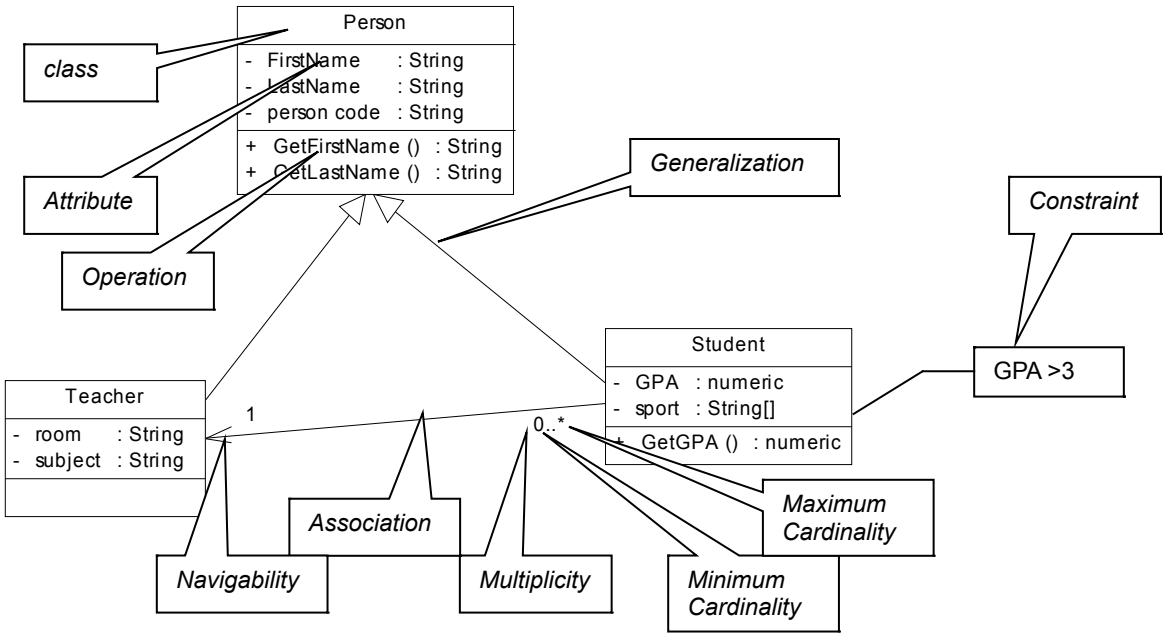
For displaying the static structure of an object oriented model UML uses class diagrams. Because of the importance of the static structure, class diagrams have more attention in the object-oriented analysis and are most in use. Because data structure are the static part of a model in this material will be treated only the class diagrams. According to a division of class diagram concepts [See the bibliography 1], there are:

- Essential part of a class diagram
- Advanced part

In essential part is used 90% in class diagrams and the rest 10%. I would like to use the same division, because in the transformation is used more the terminology of the first part and rarely of the second part.

2.1 Essential part of a class diagram

The notation used in essential part cover the main structure of the class diagram. This includes classes and relationships between them. Let's have a simple example to identify the elements of a class diagram



In the essential part of a class diagram as it is shown in picture are listed:

- Classes.
  The main structures of a class diagram
  In classes are shown also attributes and operations, which operate on attributes.

- Associations
  Associations are relationships between classes.
  In each side of an association there is a multiplicity that shows the minimum cardinality and maximum cardinality. There is also a symbol called navigability, which shows if the reference will be part of the class in the other side of association.
  Depending on minimum and maximum cardinalities multiplicities can be called:
  Multiplicity: optional if minimum cardinality is 0
  Multiplicity: multi value if maximum cardinality is *
  A combination of both possibilities.

- Generalizations
  Generalizations are special relationships between classes where inheritance is involved.

- Constraints
  Part of class diagrams are constraints that mostly are found as comments related with the class where they are relevant by a line and by using the spoken language.

2.2 The advanced part of class diagrams

The notation used in advanced part is used rarely. Often the use of this notation is needed. The symbols and concepts used in this part are very large and some of them have to do with the implementation part of the class.
Some of the notations used in this part are:

- Stereotypes
  Stereotypes can be of classes, associations or generalizations. They are called differently because they carry out more specific role than other classes, associations or generalizations. They can be seen as classes, associations or generalizations with special constraints on it.

- Classifications
  Classifications have to do with generalizations. There are single classifications and multiple classifications. It means an object can be of one subtype or more than one.

- Aggregation and Composition
  They are special kind of associations between classes.

- Derived Associations and Attributes
- Association Classes
  There are classes, which relates to a association. They are used when it is needed to have an attribute for an association.
- Visibility
  It is applied in attributes, associations, classes, etc. It has more to do with the physical implementation of the class

For the transformation, which will be treated, the second set of notations is not of very importance with some exceptions. Where it is used it will be explained.

2.3 Additional notations

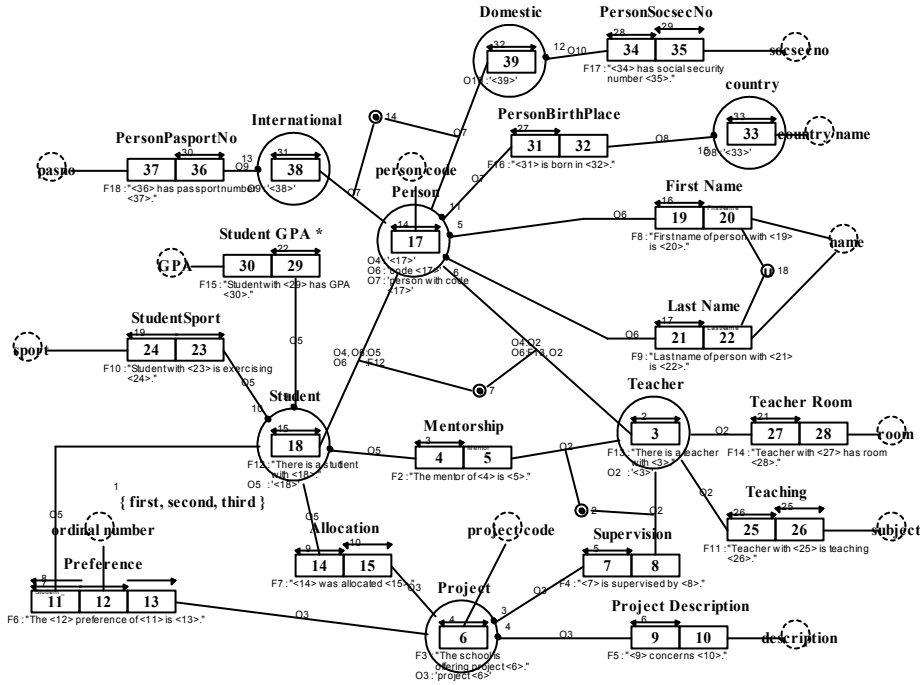
What was mentioned above is part of UML notation. For having less lose of information from the FCO-IM grammar, the above-mentioned notation is extended with some more notations. These notations are needed to cover essential parts of the data model, which mostly are not of much importance in Object Oriented Modeling and can be covered by different constraints. Because in data modeling they are a lot in use, the notation of data model is imported here. These are:

- Domains
- Identifiers

These are supported from the tool Power Designer 9.

3 The transformation of FCO-IM grammar diagram towards a class diagram

It will be shown in this part a possible way to transform an elementary information grammar diagram or EI-IGD towards a class diagram. There are several steps, which cover this transformation. To understand better the transformation and choices made, an example will be used to illustrate it. The example is about the student case extended with some other fact types. I extended to have more cases involved in the transformation. It can be the case this example is not consistent with the reality, but for our case is good enough.

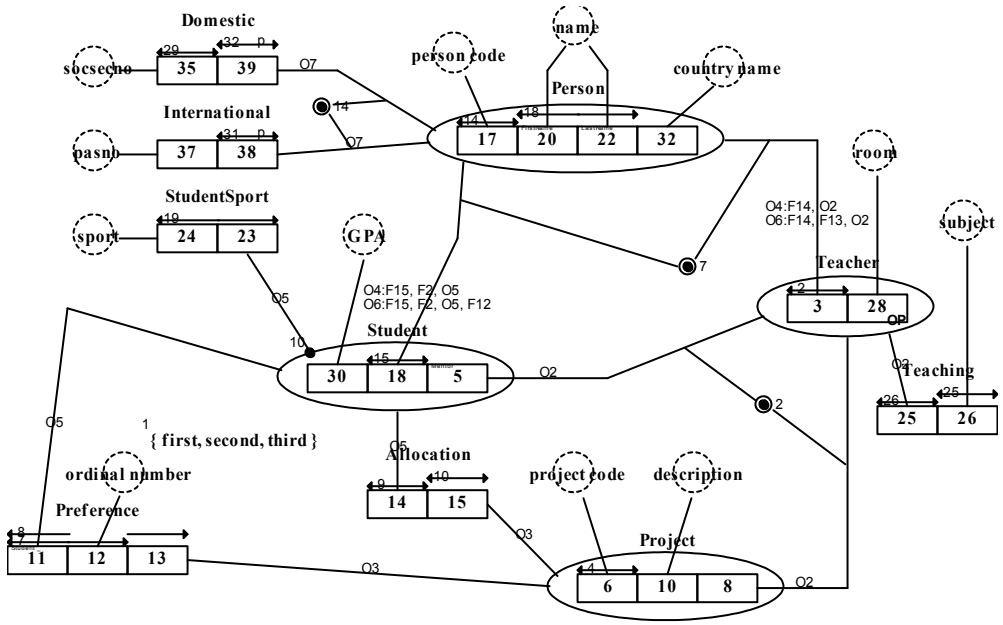


The transformation consists in 2 parts:

- Transformation within FCOIM Case Tool. Step 3.1
- Transformation of the result from FCOIM Case Tool to class diagram. Steps 3.2 until end.

3.1 (Partly) grouping and reducing the EI-IGD

This step is done within case tool and is a normal grouping and reducing process leaving out the lexicalizing (see the note in the end of paragraph). In this stage of the diagram we can identify classes and relationships between them. After this step the above EI-IGD will look like:



Fact type expressions, Object type expressions are removed from the Grouped and reduced Information Grammar Diagram or GR-IGD. Let's

explain something about this GR-IGD so it can be more clear later why I made some choices in producing this GR-IGD.
Firstly I didn't group away every role that would be grouped away using the default settings. I am talking about "Teaching" fact type. I made this choice because can be the case that we don't have the default situation always. Also the "Allocation" fact type is not reduced to have again the symmetry present between "student" and "project".
In Object Oriented Modeling this symmetry will not produce any redundant information like it is the case in Entity Relationship Modeling.
Fact type "Domestic" and "International" are not reduced because they are subtypes of fact type "Person".

Note: The transformation that will come in next steps works out also if we use the elementary EI-IGD without grouping at all, but in that case it is needed to reduce some of the lexical object types present that really are not needed by using the reduce step by step. This manual step is needed to have less classes in the final result. The result will be the same with some changes in names of some associations. If the EI-IGD is used as a source for the transformation then less loss of information is exported in class diagram. I could mention here the derived fact types, which can be identified in EI-IGD, but are lost in GR-IGD. Also some constraints derived by Totality Constraints can be lost after grouping & reducing.

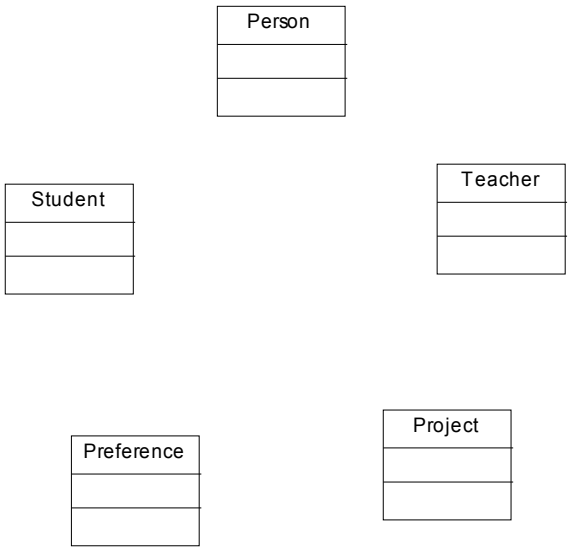
3.2 Retrieving classes

Every fact type that fulfills the below conditions will become a class in the class diagram. Each class get the name of the fact type is coming from. And visibility is PUBLIC.
Conditions are:

- It is a non-lexical object type
  Example: From the example fact types, which will become classes are: Person, Project, Student, Teacher.

Or

- Non binary fact type
  Example: From the example fact types, which will become classes are: Preference.



If we compare with entities generated in the transformation towards ERD, we see that binary fact types, which have at least one lexical role don't become class. This is because an attribute in class diagrams can have more than an atomic value. We will see in "Retrieving attributes" how these fact types will take part in transformation in the second type of attributes.

3.3 Retrieving attributes

Every lexical object type becomes an attribute. An attribute will get the name of the object type that played the role, which is played by lexical object type combined with fixes if they exist for the role. Visibility of the attributes is "PRIVATE" if we suppose to access them through operations.
There are two kinds of attributes:

- First type of attributes: Attributes coming from lexical object types that play roles that are part of fact types already become classes. These attributes have multiplicity
  - 0..1 if they play an optional role
  - 1 if they play a non optional role

  These attributes will be member of classes coming from fact types where are found roles played by these lexical object types.
  Example: Below is the table with the class and the attributes generated:

Class Name	Attribute Name	Domain	Multiplicity
Person	FirstName	name	1
Person	LastName	name	1
Person	Country	country	1
Person	Person code	person code	1
Preference	Ordinal number	ordinal	1
Project	Description	description	1
Project	Project code	project code	1
Student	GPA	GPA	1
Teacher	Room	room	0..1

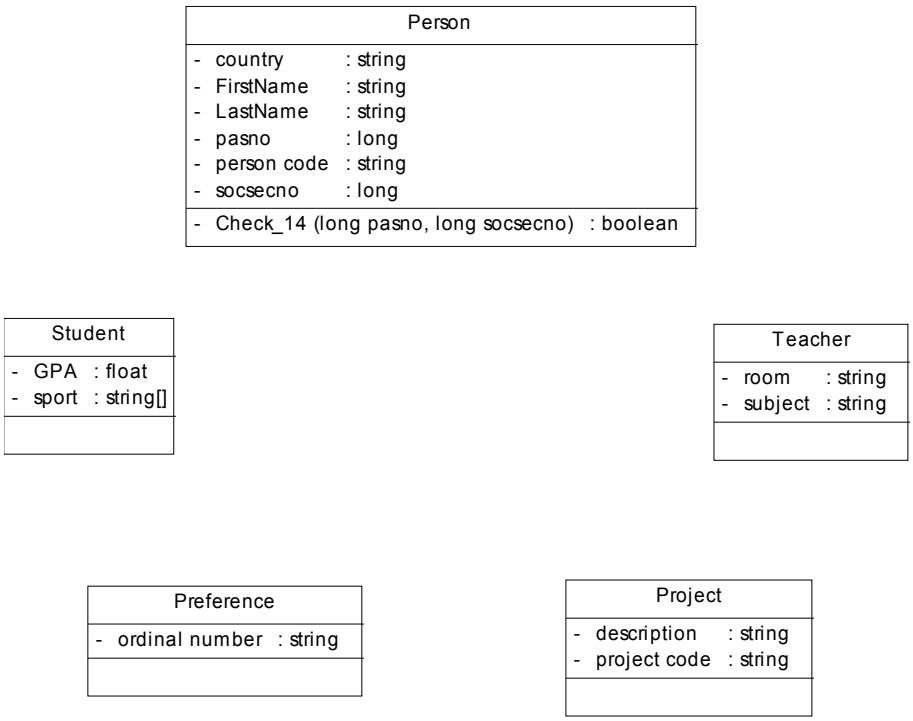
In this table are present Domains. They will be treated later. Domain is one of the notations that is not part of UML, but is imported to simplify the use of data type from the ER Modeling.

- Second type of attributes: Attributes coming from lexical object types that play roles on fact types, which couldn't become classes. These fact types are binary and have one lexical role and the other non-lexical. These attributes have multiplicity composed by their minimum and maximum cardinality:
  - If the role which is played by the lexical object type has a single role totality constraint than minimum cardinality is 1, else is 0
  - If the role which is played by the lexical object type has a single role unicity constraint than maximum cardinality is 1, else is *
  - If binary fact type is derived, attribute is marked as derived
  - If there is a totality constraint covering more than one role of non-lexical roles, and those roles are played by the same object type, a check operation (constraint) is generated as well forcing the existence of at least one non null value between attributes generated where these roles are involved. If roles are subtype role then exclusion constraint between them is involved as well. Operation name <check> + '_' + <Totality constraint> with parameters all attributes generated that are involved in Totality Constraint. Return value of operation is boolean.

  These attributes will be members of classes coming from lexical object types, which plays non-lexical role of the binary fact types.

  Example: Below is the table with the class and the attributes generated:

Class Name	Attribute Name	Domain	Minimum Card	Maximum Card	derived
Person	pasno	pasno	0	1	No
Person	socsecno	socsecno	0	1	No
Student	sport	sport	1	*	No
Teacher	subject	subject	0	1	No



3.4 Retrieving associations

There are 2 kind of associations which are retrieved.

3.4.1 Associations coming from non-lexical roles

First kind of association is coming from any non-lexical role of fact type which are become classes. Another way of saying this is: associations coming from non-lexical roles fact type which are not binary with a non-lexical role. This association will be between the class coming from the fact type where the role is and class coming from the fact type the role is played by.
The name of these kind of associations will be:
<fact type name where the role is> + '_' + <Fact type which plays the role> + '_' + <role number>.

3.4.1.1 The association side in the side of the class coming from the fact type where the role is has these properties:

- ***Multiplicity*** is combination of minimum and maximum cardinality:
  - Minimum cardinality is 1 if the role is a single role totality constraint, else is 0
  - Maximum cardinality is 1 if the role is under a single role unicity constraint, else is *
- ***Navigability*** is No
- ***Visibility*** is 'PUBLIC'
- ***Role***. It can be some short description about this relationship

3.4.1.2 The association side in the side of the class coming from the fact type which plays the role has these properties:

- ***Multiplicity*** 0..1 if the role is optional, else is 1..1
- ***Navigability*** is Yes
- ***Visibility*** is 'PUBLIC'
- ***Role***. Better not to have role in both sides

Example: Below is the table with associations of the first kind for the student case:

Association Name	Class Name1	Navigability1	multiplicity1	Class Name2	Navigability2	multiplicity2
Preference_Project_13	Preference	No	0..*	Project	Yes	1..1
Preference_Student_11	Preference	No	0..*	Student	Yes	1 1
Project_Teacher_8	Project	No	0..*	Teacher	Yes	1 1
Student_Teacher_5	Student	No	0..*	Teacher	Yes	1 1

3.4.2 Associations coming from binary fact types

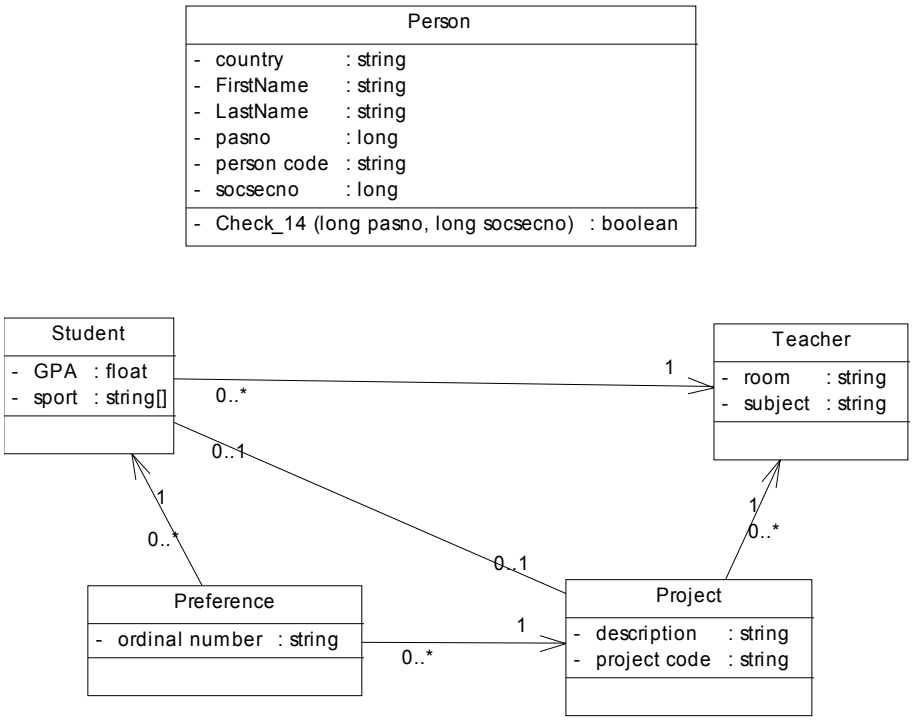
These associations are coming from binary fact types with both roles non-lexicalized.
Both sides are treated in the same way.

- ***Multiplicity*** is combination of minimum and maximum cardinality:
  - Minimum cardinality in the side of the entity type coming from the non-lexical object type which plays the role is 1 if there is a single role TC on the other role, else 0
  - Maximum cardinality in the side of the entity type coming from the non-lexical object type which plays the role is 1 if there is a single role UC on the other role, else n
- ***Navigability*** is Yes. In this way the symetry is still present
- ***Visibility*** is 'PUBLIC'
- ***Role***. It can be some short description about this relationship in of the sides.

Example: Below is the table with associations of the first kind for the student case:

Association Name	Class Name1	Navigability1	multiplicity1	Class Name2	Navigability2	multiplicity2
Allocation	Student	Yes	0..1	Project	Yes	0..1

There are some additional properties of the associations.
One of them is aggregation or composition. It can be assigned to association side if the semantics are consulted in the diagram. If there are some fact expressions like is composed or has than perhaps that association has an aggregation into that.



3.4.3 Another option for associations

The second type of associations the definition can be more general and to say every fact type which more than one non-lexical role must be an association between lexical object types playing those roles. From these fact type are generated also classes of the second type (see in retrieving classes-Non binary fact type). In this case the generated class would be an associations class for the generated association.
Note: In the tool is not implemented this kind of association. In Power Designer associations can have only two sides and by the definition above associations can have more than 2 sides. In Internal Repository tool both kind of associations can be supported.

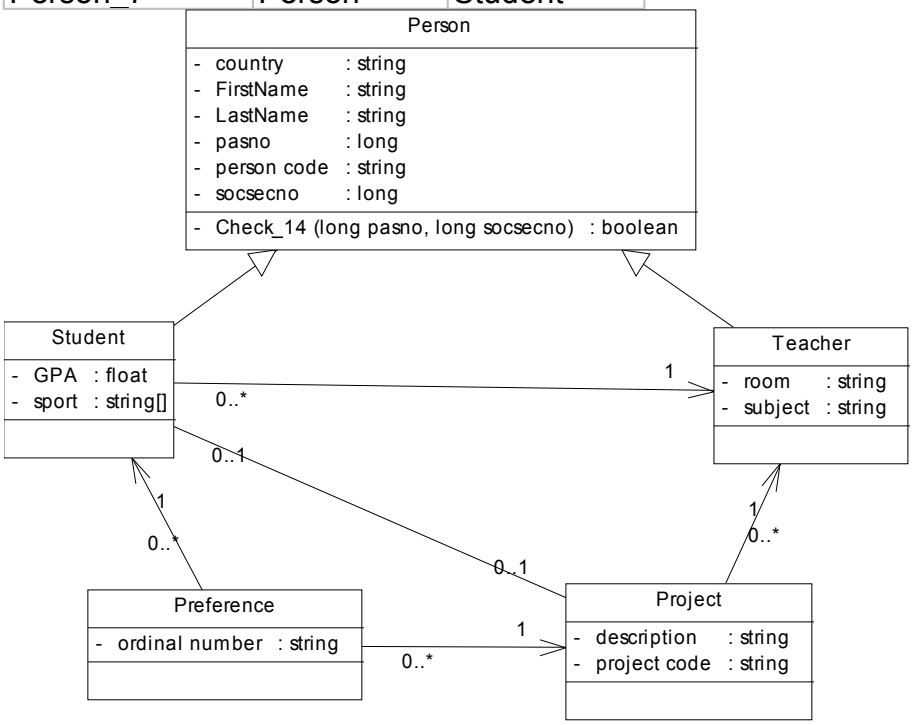
3.5 Retrieving generalizations

Generalizations are a special kind of relationship between classes. In FCOIM they are called subtypes or better to say super types. These notations are also present in UML with the same name a meaning. So part of a generalization relationship are subtypes and super types. Generalizations are retrieved by subtypes in GR-IGD. A generalization has a super type the super type in GR-IGD and as subtype the subtypes. More than one generalization can be present for a super type. If roles played by a super type are involved in the same totality constraint than they are part of the same generalization. If no

totality constraint is present than there is only one generalization. The generalizations coming from the GR-IGD are by default complete and exclusive because they are derived or declarative.
The name of a generalization is <fact type which acts as a super type class> + '_' + <if exists totality constraint number>

Example: Below is the table with generalizations of the student case:

Generalization	Parent	Child
Person_7	Person	Teacher
Person_7	Person	Student



3.6 Retrieving domains

The notation 'Domains' is not part of UML notation, but is imported by ERM. It introduced in class diagram because it can better manage some constraints in domain level.
In this transformation domains are retrieved by lexical object types. Also are every fact type which was become a class will be a domain. This is to simplify the use of data types. In the list of domain are included special domains 'void', 'boolean', which are not present at all in GR-IGD, but are really needed in Object Oriented Modeling.
Domains get the name of the lexical object type they come from.
Example: Below is the table with domains of the student case:

Name	Type	Length	Scale
Void	Void		
Boolean	Boolean		
Subject	String	3	0
Sport	String	4	0
Room	String	4	0
Project code	String	4	0
Person code	String	3	0
Ordinal	String	6	0

Name	String	6	0
Description	String	44	0
Country	String	15	0
Socsecno	Long	6	0
Pasno	Long	7	0
GPA	Float	2	1
Teacher	CLASS		
Student	CLASS		
Project	CLASS		
Person	CLASS		
International	CLASS		
Domestic	CLASS		
Preference	CLASS		

3.7 Retrieving identifiers

Identifier is the second notation introduced in UML that is not part of it. This is important for data models, but in Object Oriented not really. Important constraints are covered using this notation. This is the reason why is introduced also here.
An identifier is a class attribute or an association side where the class is involved, or a combination of class attributes and association sides where the class is involved, whose values uniquely identify each occurrence of the class. Each class can have many identifiers. Among identifiers, the primary identifier is the main identifier of the class. As you can find out by yourself now identifiers are retrieved by Unicity Constraints.
Identifier will become every Unicity Constraint that cover roles which are played by lexical object types or by roles involved in associations of the first kind.
Note1: In Power Designer only identifiers where only attributes are involved are present. Association sides cannot be present in an identifier. So the Unicity Constraints over the roles, which are involved in any association will be lost.
Note 2: In generated VB Script Identifiers are not included. They are present in Internal Tool Repository.

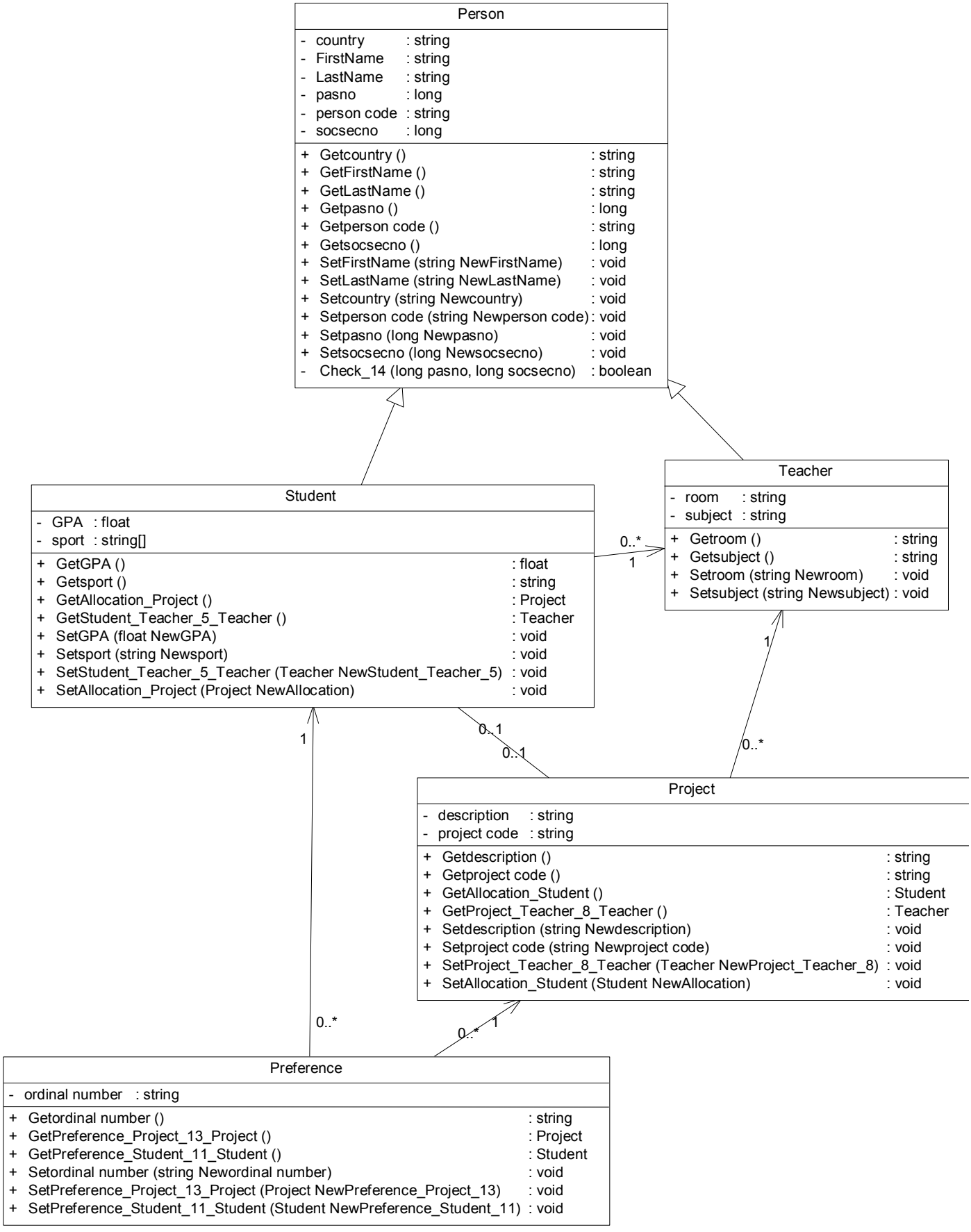
3.8 Retrieving operations

For every attribute and association side are generated 2 operations for getting and setting the value of them.

3.9 Retrieving other constraints

Using domains and identifiers helps to define the main constraints present in GR-IGD. For other constraints like subset constraint, cardinality constraint, exclusive constraint a constraint can be written down with a line related to the class where this constraint is applied.

The final class diagram is:


Person
- country : string
- FirstName : string
- LastName : string
- pasno : long
- person code : string
- socsecno : long
+ Getcountry () : string
+ GetFirstName () : string
+ GetLastName () : string
+ Getpasno () : long
+ Getperson code () : string
+ Getsocsecno () : long
+ SetFirstName (string NewFirstName) : void
+ SetLastName (string NewLastName) : void
+ Setcountry (string Newcountry) : void
+ Setperson code (string Newperson code) : void
+ Setpasno (long Newpasno) : void
+ Setsocsecno (long Newsocsecno) : void
- Check_14 (long pasno, long socsecno) : boolean
Student
- GPA : float
- sport : string[]
+ GetGPA () : float
+ Getsport () : string
+ GetAllocation_Project () : Project
+ GetStudent_Teacher_5_Teacher () : Teacher
+ SetGPA (float NewGPA) : void
+ Setsport (string Newsport) : void
+ SetStudent_Teacher_5_Teacher (Teacher NewStudent_Teacher_5) : void
+ SetAllocation_Project (Project NewAllocation) : void
Teacher
- room : string
- subject : string
+ Getroom () : string
+ Getsubject () : string
+ Setroom (string Newroom) : void
+ Setsubject (string Newsubject) : void
0..*
1
1
0..*
1
0..1
0..1
Project
- description : string
- project code : string
+ Getdescription () : string
+ Getproject code () : string
+ GetAllocation_Student () : Student
+ GetProject_Teacher_8_Teacher () : Teacher
+ Setdescription (string Newdescription) : void
+ Setproject code (string Newproject code) : void
+ SetProject_Teacher_8_Teacher (Teacher NewProject_Teacher_8) : void
+ SetAllocation_Student (Student NewAllocation) : void
0..*
0..*
1
Preference
- ordinal number : string
+ Getordinal number () : string
+ GetPreference_Project_13_Project () : Project
+ GetPreference_Student_11_Student () : Student
+ Setordinal number (string Newordinal number) : void
+ SetPreference_Project_13_Project (Project NewPreference_Project_13) : void
+ SetPreference_Student_11_Student (Student NewPreference_Student_11) : void

4 Implementation of the prototype

FCO-IM Case Tool offers the opportunity to export the repository in dBase relation format. Using this great opportunity is it possible to access the EI-IGD or GR-IGD in a nice way. Based on this repository and in the transformation above, a prototype, which supports this transformation is present.

4.1 Implementation platform

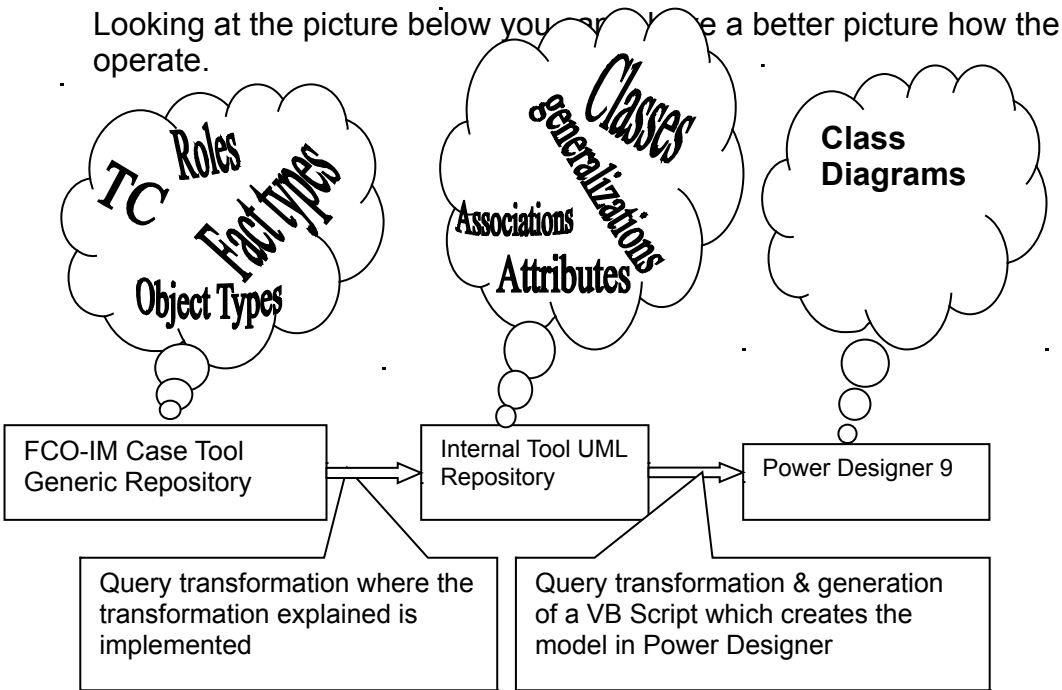
The tool is implemented in MS Access 97. The reason is because it offers in the same time an environment for operating with relational databases (in this case with dBase format) and a fast interface builder, which makes it a good tool for prototypes. Also the size of the database in use is relatively small in structure and possible population.

4.2 Destination platform

As a destination platform to showing the final result coming from the tool is chosen Power Designer 9. This version of Power Designer supports both Data Oriented Models and Object Oriented Models. There are also possible the conversion from data model to class diagrams within Power Designer, but I don't know if it is reliable. There is also a repository where can be stored every kind of model produced with Power Designer. This version is using XML technology for the storage of their documents. It offers also the opportunity to run VB Scripts. Using this script it is easy to create models of different diagrams including here class diagrams. This last mentioned feature is used for producing the final diagram in Power Designer. Using Power Designer it is possible the creation of classes in different platforms like C++, Java, different XML dialects, etc. by class diagrams. Beside this it offers also the necessary documentation to operate with all these features.

4.3 The structure of the tool

Looking at the picture below you can have a better picture how the tool operate.



4.4 FCO-IM Case Tool Generic Repository

For accessing the repository of FCO-IM grammar diagram, FCO-IM Case Tool offers a module for exporting that repository in dBase IV format. That makes possible to use against this repository SQL.
Rules defined in transformation above can be easly expressed with SQL.

4.5 Internal Tool Repository

Within the tool there is a database where the information about the meta data of the UML model is stored. This repository stores the basics of a class diagram and some of the advanced elements.
The analysis for this model were performed with FCO-IM Case Tool. The reason was good documentation while doing analysis and building the model.
This repository was needed, because:

- it was more clear what is needed after the data model was built
- it is a static source of data for transforming the data towards a platform like Power Designer 9
- it would be easier for future transformation towards other platforms
- bugs can be identified more easily
- the result after transformation can be consulted more easily

The data model is found in appendix 1.

4.6 Query transformation from FCO-IM exported Repository towards Internal Tool Repository

All rules explained in the transformation for retrieving classes, attributes, associations, domains & value constraints, generalizations, identifiers, operations are implemented as views/queries in MS Access.
After that with insert queries against these views/queries, data was pumped from FCO-IM Repository to Internal Tool Repository.

For these rules applied using SQL you can check the appendix 2.

4.7 Query transformation from Internal Tool Repository and generation of VB Script for Power Designer 9

These query transformation are queries/views against the Internal Tool Repository. These queries/views are specific for the creation of VB Script for Power Designer 9.

For these queries/views are see appendix 3.

4.8 Display module

Within the tool there is a display module composed by a form and a module. It is built to consult the UML class diagram stored in Internal Tool Repository. For displaying the result list boxes are used.

5 Bibliography

♦ Modeling OO databases with FCO-IM
Report Chiel Boon
♦ UML Data Models from ORM Perspective
Papers by Dr. Terry Halpin
♦ FCO-IM and ERM
Paper by Guido Bakema
♦ UML Distilled
Fowler & Scott
♦ Applying UML and Patterns
Craig Larman