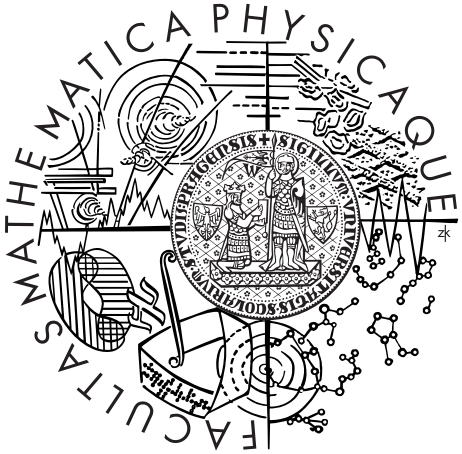
Charles University in Prague
Faculty of Mathematics and Physics

MASTER THESIS



Bc. Petr Sobotka

Transformace z OCL do SQL

(Transformation from OCL into SQL)

Department of Software Engineering

Supervisor of the master thesis: Doc. Ing. Karel Richta, CSc.

Study programme: Computer Science

Specialization: Software Systems

Prague 2012

Many thanks belong to my supervisor, Doc. Ing. Karel Richta, CSc. for leading me through all the obstacles that came up during the making of this work, to my family for the shown patience and support, and to Sparx Systems and Microsoft for providing the software used.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In ..... date ..... signature of the author

Název práce: Transformace z OCL do SQL

Autor: Bc. Petr Sobotka

Katedra: Katedra softwarového inženýrství

Vedoucí diplomové práce: doc. Ing. Karel Richta, CSc., Katedra softwarového inženýrství

Abstrakt: Cílem této diplomové práce nazvané Transformace z OCL do SQL je prozkoumat možnosti rozšíření CASE nástroje Enterprise Architect o schopnost generování SQL kódu, který implementuje integritní omezení v OCL z platformově nezávislého modelu, a vytvořit funkční prototyp, který tuto schopnost demonstruje. Práce obsahuje stručný úvod do problému modelem řízeného vývoje, proč by bylo užitečné zavést specifikovaná integritní omezení přímo v databázi, popis interakce s nástrojem Enterprise Architect spolu s analýzou, jak se dostat k potřebným datům, způsob, jakým může být OCL mapováno na SQL, a nakonec popisuje implementaci zásuvného modulu OCLtoSQL od analýzy a zvolené architektury až po jeho dokumentaci.

Klíčová slova: UML, OCL, EA, SQL

Title: Transformation from OCL into SQL

Author: Bc. Petr Sobotka

Department: Department of Software Engineering

Supervisor: doc. Ing. Karel Richta, CSc., Department of Software Engineering

Abstract: The aim of this Master Thesis named Transformation from OCL into SQL is to explore possibilities of enhancing Enterprise Architect by the ability of generating SQL code that implements OCL constraints of a Platform Independent Model, and to create a working prototype that demonstrates this capability. The thesis contains a brief introduction to the problem of Model Driven Architecture, why it would be useful to implement the specified constraints directly in the database, a description of the interaction with Enterprise Architect along with analysing how to get the needed data, the way OCL can be mapped onto SQL, and finally it describes the implementation of the OCLtoSQL plugin from its analysis and chosen architecture to its documentation.

Keywords: UML, OCL, EA, SQL

Contents

Introduction

OCL

Unified Modeling Language (UML, OMG standard since 1997) is a formal language that can be used to express specifications from various views by defining comprehensible diagrams that may be displayed by many tools. However, UML diagrams alone are not able to express all the nuances needed in complete specifications, such as advanced business rules. In order to do so, another instrument is needed. When UML standard was being created, there existed several possibilities, but in the end it was OCL that was added as a part of UML standard.

Object Constraint Language (OCL) is a declarative, strongly typed language that augments UML (along with other object describing standards) and enables it to make the specifications more precise. It is simple, compact and uses no advanced mathematical symbols while having enough power to add any (sensible) rules applied to the models. Being object oriented, it is easy to use for anyone with some programming experience.

OCL has roots in an expression language from Syntropy method and its offspring, IBM business modeling language. Nowadays, OCL is widely used as a language that may describe details about objects and their supposed behaviour. Part of the language are Boolean expressions that can constitute the constraints, arguably the most often used feature of OCL in the object-oriented modeling.

Why OCL to SQL

In the work of a software developer there exist many activities that are both mentally exhausting and time-consuming. Also very often they are done repeatedly. Therefore some sort of automation is very welcome. Getting rid of repetitive work saves both time and nerves, making the developer more productive and less mistake-prone.

One form of possible automation is code generation. By generating parts of code, one has less places where they might make mistakes (and, as already mentioned, is more efficient). However, it is generally not a good idea to just assume what the programmer may have thought by the particular part of code. It is better to just use the easy, straightforward parts as an input, rather than forcing the users to backtrack and look up all details of how exactly the employed tool works in some not often shown situation. And either way, often the hardest to find mistakes are in some basic parts of code, the code generation when used in such places prevents them. All in all, every bit of saved work can help if done transparently and the developer can know exactly what is going on when he uses it.

While typically code is generated from some other code and metadata, why not take it a step further and generate it already from specification? Of course in order to do so, the specification would have to contain all needed information (be complete) and be described in a formal way that could be used as an input for further transformations and generations (be written in strict, exact and powerful enough languages - for example UML and OCL). A set of complete and

formal parts of specification that describe the system from various viewpoints is called model. Model Driven Architecture (MDA) is an approach of software engineering that uses models as a specification and enables to transform models with higher level of abstraction to less abstract models describing the same, and ideally generate code implementing them. While the MDA's emphasis is on forward engineering (from a higher level abstraction to a lower one), the other direction is used too. That way any changes in lower levels of abstraction can be automatically projected upwards or a previously unspecified system may be analyzed and/or its specification created etc.

Models that describe the requirements and use cases (the what, but not how) are called Computation Independent Models (CIM). Models containing no information how to implement the system, but only necessary services and data are Platform Independent Models (PIM). And models that map what was said in PIM on particular platform (ie. Java, C#/.NET, some sort of database) are called Platform Specific Models (PSM). The last level is the implementation itself - Implementation Specific Model (ISM).

Using the aim of this work as an example, one could describe classes and business rules as a PIM in UML and OCL, use some tool to transform this model into PSM. This PSM could describe tables, their columns of appropriate types and foreign keys between the tables (according to the associations from PIM) in DDL from which it is easy to produce SQL code creating the same structures in a database. Business rules that were in PIM (but are not part of PSM), could then be used to generate SQL code directly. This way one can automatically create whole database with the needed safeguards directly from its specification.

Another question one might have is why to have constraints already in the database when it would probably be easier and often times quicker to implement them just in some of the higher layers? One of the reason is that sometimes the business rules deal with not only the data accessible in the current context and layer, but also with the data that have to be somehow queried from lower layers - and just getting them for every simple check might rise the data transfer requirements inappropriately. Instead the data can be already checked in the database and thus do not have to be transferred. Also, in practise various layers might be used by more than one system and it would be a lot harder to manage constraints separately, rather than keeping them in one place. For example one database may serve as a data storage for several applications that need to easily use each other's data and in that case it would be easier and efficient to have as much as possible constraints implemented in the shared database itself.

Either way, implementing some of the constraints in the database may be very useful in many real-life situations and being able to automatically generate SQL code directly from specification can save both time and possible mistakes.

Existing Solutions

There are not many solutions that are able to generate SQL code implementing OCL constraints directly from an UML model. At least not free ones.

Arguably one of the most extensive work that has been done in this area of interest is in the Dresden OCL Toolkit (DOT) Java project, developed and maintained by Software Technology Group at Technische Universitat Dresden

since 1999. In 2005 Dresden OCL Toolkit 2 (DOT2) was released, part of which is the OCL22SQL tool that enables SQL generation from OCL2.

OCL22SQL is able to generate the database schema according to the classes in the model and their associations, and to produce views for each OCL constraint that return data on which the constraint does not hold. It also generates views over each table that hide the way the possible generalization issue was solved (one may choose if by using one table also containing all columns used in its children, or by separate tables connected by foreign keys).

OCL22SQL supports only Postgress, Oracle and Standard SQL dialects (according to [12]). Being written in Java, it is not very compatible for integration into .NET based environment. The tool is also not able to get the OCL rules directly from the model, but requires to have them in a separate file - so one has to provide both .XMI and .OCL files as input. Because OCL22SQL generates not only the constraints checking code, but also the whole DB schema from the classes, it is not suitable for augmenting CASE tools that already have their own DDL code generation (it may have been used by different tools and/or plugins, so changing anything could be out of question). OCL22SQL only supports view generation from OCL constraints which can be used for asynchronous validating and are more economical in the department of performance (the checks may be done only when wanted and not every time data are inserted or modified), however the data in the database may at a given time violate the constraints and so the views are not always the right solution (as opposed to for example database triggers).

Very useful in finding out more about available MDA tools generating code from OCL/UML that are also worth mentioning was [7] where the full list along with links to their home pages may be found. Project Poseidon offers only simple Java generation from UML, does not support OCL and in the database can create only primary keys. Rational Rose (from IBM), while having about the same power in Java generation as Poseidon, has the ability to add some simple constraints via properties of the attributes, however OCL is not used in the generation in any way. MagicDraw can partly transform a PIM into a database PSM which then has be manually enriched by constraints before the SQL code may be generated, but again - generation from OCL is omitted. ObjectEngineering/UML does not support OCL, but is able to generate code both in Java and SQL that is able to enforce cardinality of any size - also interesting is its usage of database triggers in order to do so. Together (from Borland) is a tool which has SQL code generation power similar to Rational Rose, but full support of OCL for Java. Executable UML differs from others, because it generates as its name implies directly executable code, however its usage is mainly limited to real-time and embedded domains and has no support for generating constraints into the SQL code. Also not capable of portraying constraints into SQL are following tools that, while not being important enough for the purpose of this work to provide details, are listed for the sake of completeness: Octopus, OCLE, KMF, OCL2J, OCL4Java. But all in all none of these tools cover the OCL constraints from a model to SQL problem as thoroughly as the Dresden one.

Aims of this work

The main aim of this work is to describe the problem of OCL to SQL generation and its context, analyze it, and then design and implement a solution that could generate SQL code from OCL as a part of Enterprise Architect.

This solution is not intended as a direct competition to the Dresden OCL Toolkit project - it would be a bit conceited to to think that a single project of one person can overcome such a large project on which many people were working over quite a large period of time. However, as already mentioned, the OCL22SQL tool from DOT2 is not very well suited for direct usage inside EA and some of its features might be done differently. And so this work tries to bring something hopefully new, taken from slightly different angle, and most importantly be in practice more easily usable and integrable as a part of EA.

Of course in order to do so, some research of the EA's capabilities and environment has to be made - for example how to get to OCL (and other needed data) directly from selected models in the EA projects.

Also this work does not intend to do a complete transformation, just the important parts for easy and practical use. Due to the extensiveness of the assignment and the complexity of integrating the solution into EA it probably even would not be achievable in the allotted time. Therefore it just tries to show a working, albeit a simplified, way it may be done so that possible future projects may continue where it left or at least use the knowledge revealed.

Implementation is targeted on T-SQL and Microsoft SQL Server, different dialects of SQL may be implemented later.

1. Enterprise Architect

Enterprise Architect is an UML and business process modeling tool from Sparx Systems. It implements the principles of Model Driven Architecture (MDA) development and aspires to be "MDA compliant" as described in [19]. To further describe EA's functionality using a quote from [10]: "Enterprise Architect is an intuitive, flexible and powerful UML analysis and design tool for building robust and maintainable software. From requirements gathering, through analysis, modeling, implementation and testing to deployment and maintenance, Enterprise Architect is a fast, feature-rich, multi-user UML modeling tool, driving the long-term success of your software project."

For the purpose of this work, it is important that EA is able to generate DDL model from the platform independent model containing entities along with their attributes and associations as can be seen in the figure 1.1. In the DDL package the entities from PIM package (Flight, Person) are represented as tables, the entities' attributes as columns of corresponding types and associations as combination of additional tables (Attending, BeingCrew) and foreign keys.

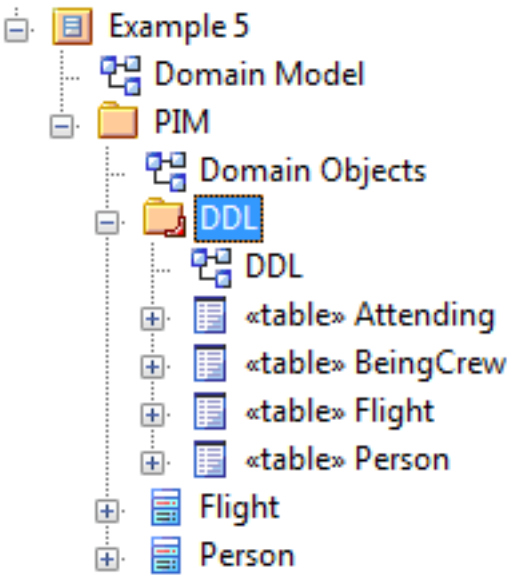


Figure 1.1: EA: PIM package with the corresponding DDL package

And from the DDL model it is able to generate SQL code of the chosen dialect (ie. T-SQL, Oracle...). Such a code is able to create entire database containing all the information from the DDL package. However in the platform independent model there may also be some OCL constraints. They (despite them being tested by EA for correctness and possibly validated) are not included in the SQL code generation. Therefore a way must be found to somehow add that ability to EA.

1.1 Plugins

Enterprise Architect's functionality may be enhanced by several ways - be it using external application separately (and synchronizing the code), calling external

application from EA, running scripts in EA or adding plugins. Plugins (or Add-Ins as they are called in EA) are arguably the most sophisticated method out of them, because they can use not only what is stored in the XMI file underneath, but also other information directly from EA - for example what package or element is selected. Being integrated into the EA IDE also makes the added functionality more comfortable to use (Add-Ins are accessible both from the Menu and the Context Menu). For communication EA exposes the Automation Interface that enables to interact with Add-ins, scripts and its objects.

One of the languages in which Add-Ins may be implemented is C#. EA Add-In is an ActiveX COM DLL that implements methods that let it react to several events that occur in the EA, like for example filling the Menu and how to react when each of the Menu items is selected. As shown in the figure 1.2, the interface and placement of the Addin DLL is stored in the COM codebase while the information of their existence is written in the Windows registry. And so EA knows from the entries in the Windows registry what Add-Ins are available and what are their names and the classes in them implementing the needed methods to be called on events of EA. If the user decides to call an Add-In, EA has its interface and location from COM codebase and so may call its methods for the events that occur. For example in the case of clicking a menu item of the Add-In with an object (maybe some package) selected in the Project Browser, method EA_MenuClick is called with the selected object's reference as one of its parameters.

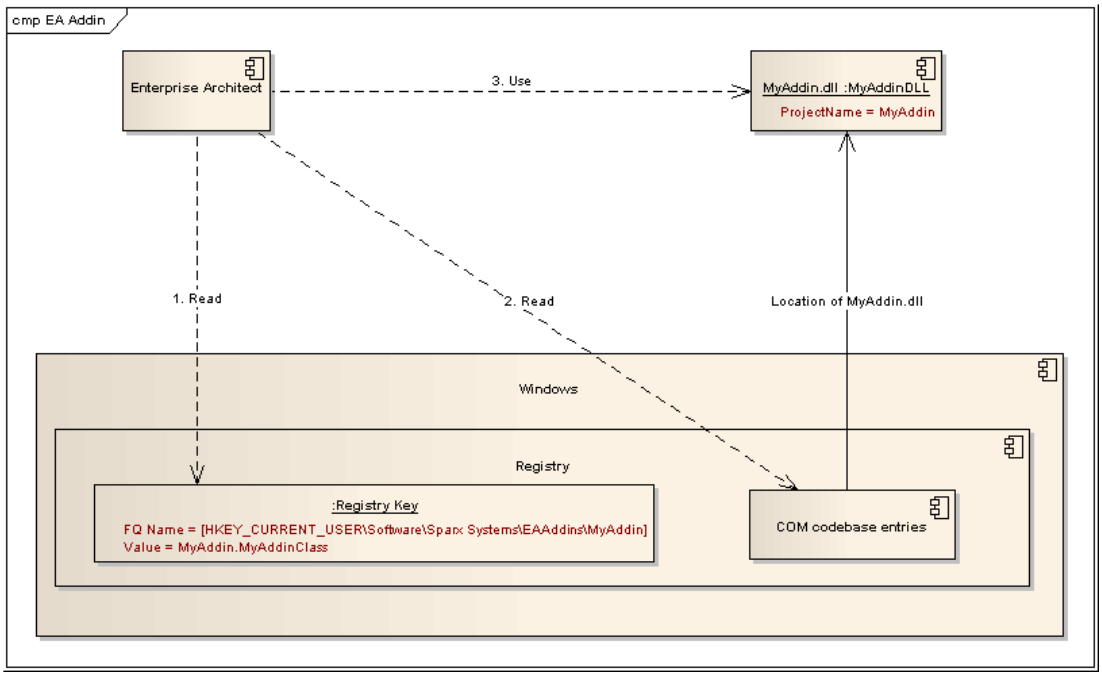


Figure 1.2: How EA Add-Ins work, taken from [20]

Following steps should be done when creating an EA Add-In in Visual Studio 2010 (mainly from [20]):

1. Create a C# DLL project
2. right-click on References, Add reference/Browse: select "EA Install/ Interop.EA.DLL"

3. Project Properties/Application/Assembly Information: tick "Make assembly COM-Visible" (figure 1.3)

4. Project Properties/Build and tick "Register for COM interop" (figure 1.4)

5. Add key (Default) with value "Project.MainClass" at the registry location "HKEY_CURRENT_USER/Software/Sparx Systems/EAAddins"

   - (ie. via regedit as shown in figure 1.6)

6. The main class (that is specified in the registry) should implement some of these methods (depending on what events it needs to react to):

   (a) EA_GetMenuItems that populates the Menu and its Submenus (and Context Menu)

   (b) EA_MenuClick that implements reactions to what is selected from the Menu (or Context Menu)

   (c) ...

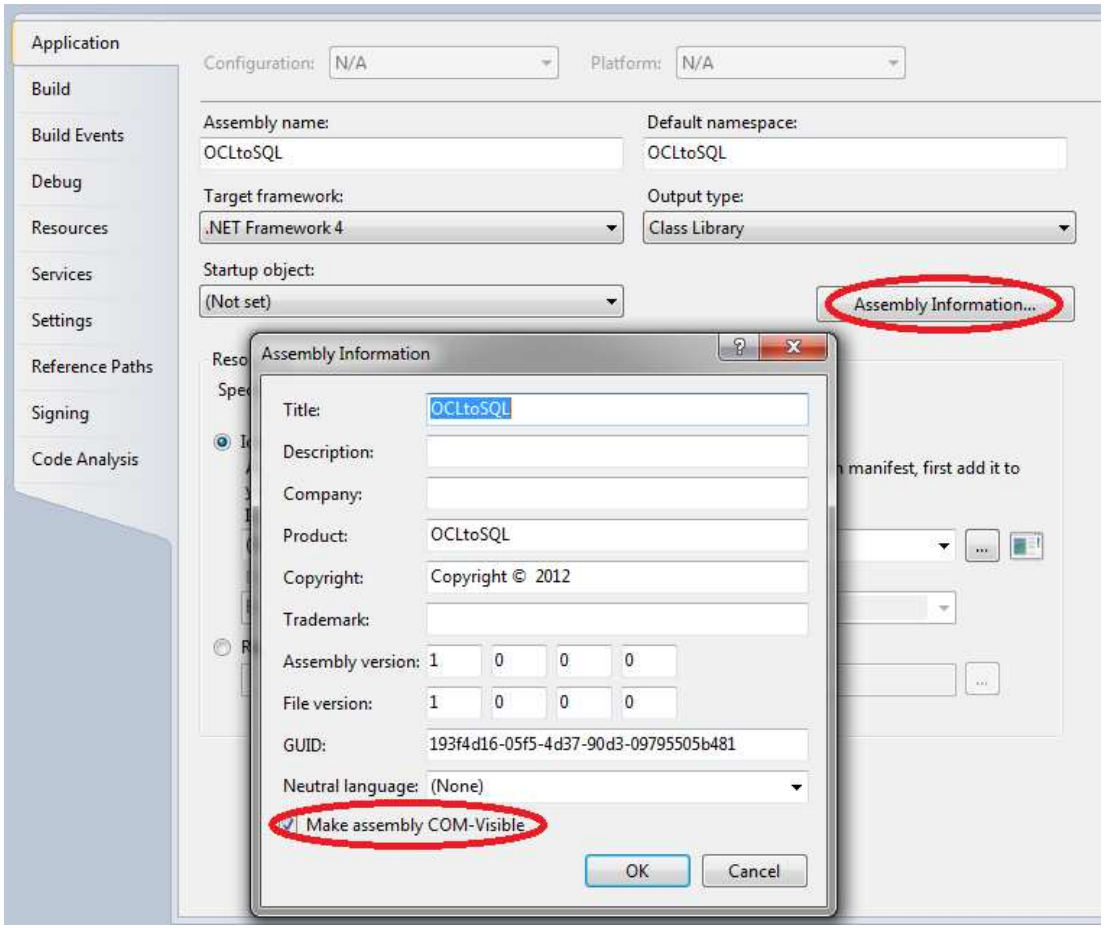


Figure 1.3: Make DLL COM visible in VS 2010

Each build will then make the new version of Add-In directly accessible in EA (in Menu and Context Menu under "Extensions", or Add-Ins in the older EA). Because of the need to register the COM object, VS needs to be "Run as administrator" in Windows 7.

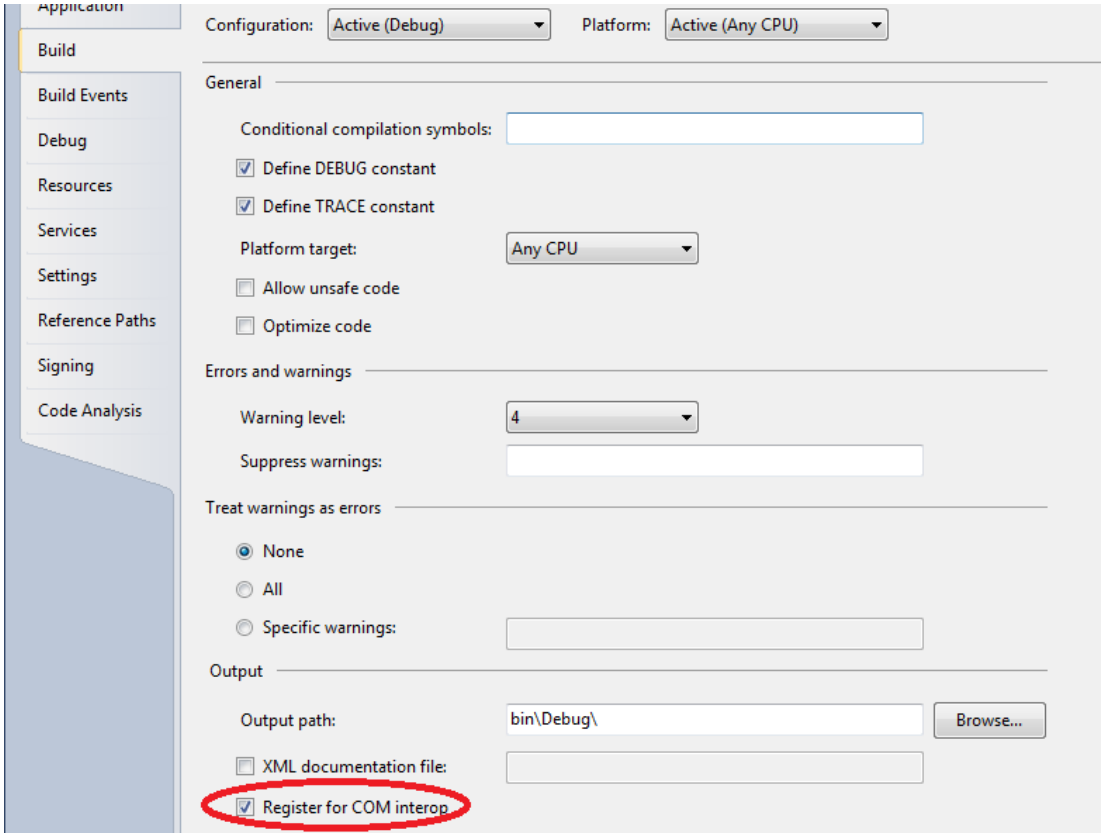


Figure 1.4: Register DLL for COM interoperability in VS 2010

1.2 Debugging Add-Ins in VS 2010

In order to debug an EA Add-In through Visual Studio 2010 (after it is built and published as mentioned in the Plugins section), EA should be run first to which then one can attach VS (Debug/Attach to Process, select EA.exe as shown in figure 1.5). Now the breakpoints and tracing through the code may be used as needed while calling the Add-In in EA. Unfortunately the attributes of EA objects cannot be directly explored (they are accessed remotely, and so the debugger does not see them unless they are accessed in the code), however tracing enables finding out where the problem occurs and then by modifying the code their values may be exposed. For example when using an element Person, the debugger sees it, but its attribute Age not, but if the code is changed to contain int x = Person.Age, the value of Age can be seen by the debugger in x.

It is important to note that due to some quirk in VS 2010, running EA automatically by Debug usually does not enable debugging it (the auto attaching of the EA process seems to fail). Probably the easiest workaround is to run EA externally and then attaching it from VS manually (as already mentioned). Also because of the need to attach to the EA process, only Professional and above versions of VS 2010 are able to debug EA Add-Ins that way.

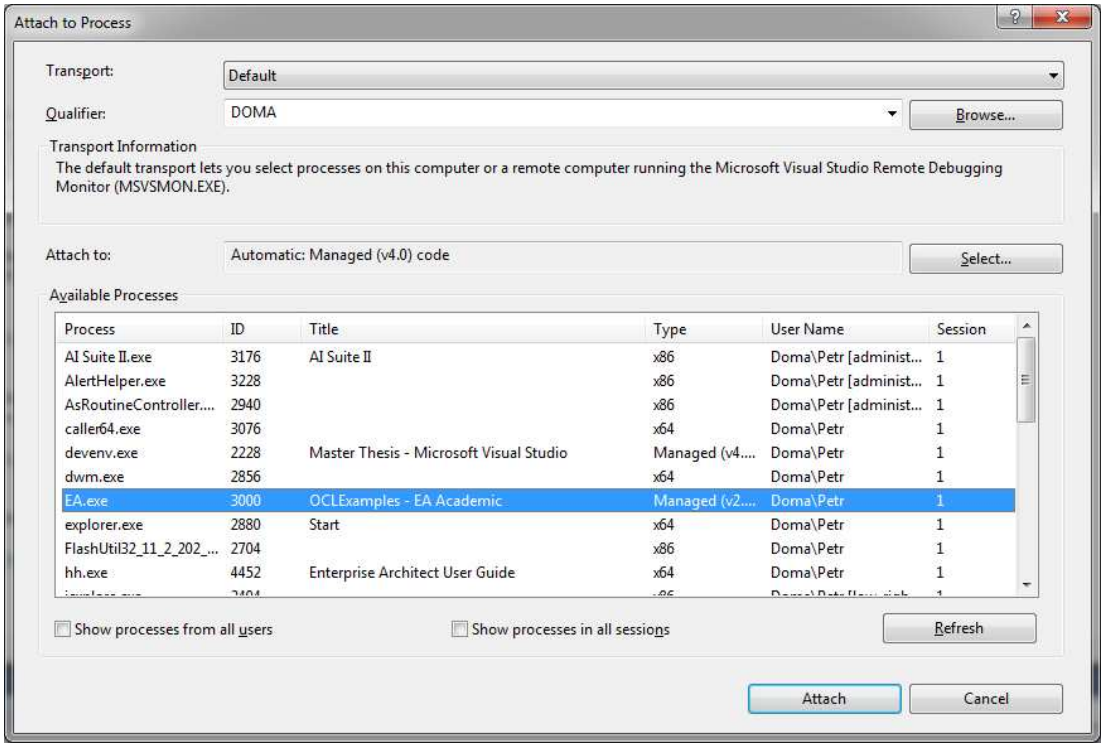


Figure 1.5: Attach to the process in VS 2010

1.3 Deploying EA Add-Ins

Assuming one has a working EA Add-In DLL (and other needed files if there are any) at the intended place, they have to do the following to install it on their computer:

1. Add key (Default) with value "Project.MainClass" at the registry location "HKEY_CURRENT_USER/Software/Sparx Systems/EAAddins"
    - (ie. via regedit as shown in figure 1.6)
2. Register the Add-In DLL for COM interoperability with the path it was copied to
    - Easiest way is probably using regasm (a tool from .NET Framework)

All of it may be of course done by an installer created for that purpose, thus making the installation of the new Add-In easy for anyone.

1.4 EA Object Model

Enterprise Architect's documentation [10] is very extensive (over 2000 pages), so unless one knows exactly what they are looking for and the correct nomenclature, it is very hard to find anything. In here only a small part of the EA's object model (which is also not small) that is directly involved in the creation of the wanted Add-In is revealed.

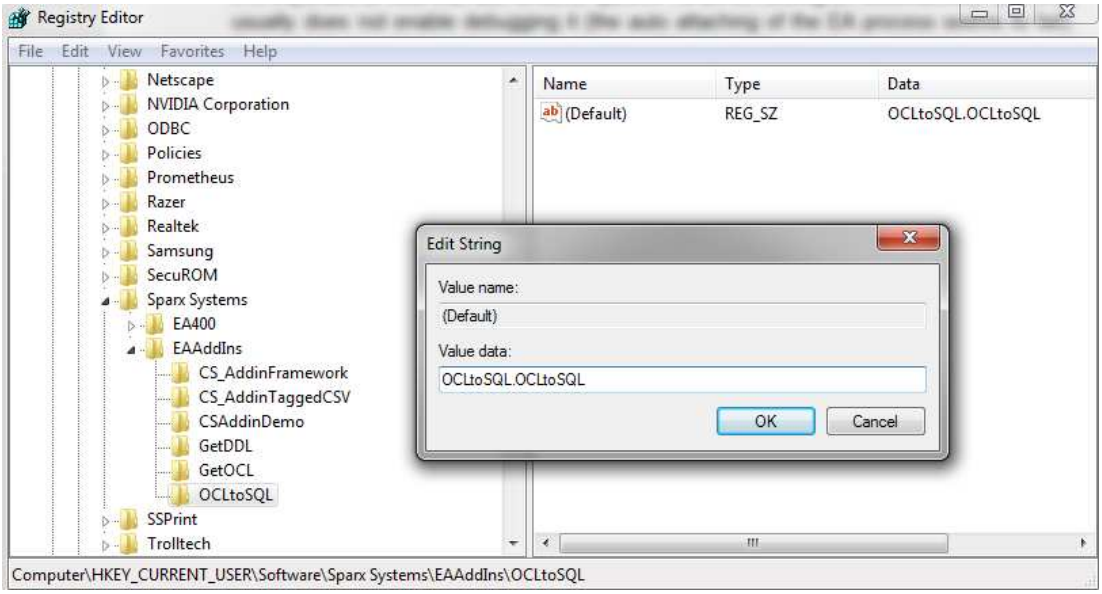


Figure 1.6: Editing Windows Refistry via RegEdit

In each method, an Add-In has to implement in order to react to the corresponding event, there is an EA.Repository reference (plus some other information depending on the method - like what item menu was clicked etc.).

Through EA.Repository one may access any part of Project Hierarchy (such as packages, elements, associations, etc.) and modify them. The repository may be searched both as a tree or by various methods getting IDs or names as parameters.

EA.Project, gained by the Repository.GetProjectInterface() method, enables importing or exporting the objects in its Repository, as well as calling transformations on them (for example PIM to DDL).

Collections in EA (represented by EA.Collection) can be iterated by foreach, directly accessed (GetAt(index)), added to (AddNew), deleted from (Delete(index)), searched by name (GetByName) or ID (GetByID) and so on. It is possible because the methods use the relatively new C# type dynamic (the real type decided during compilation).

EA.Package is similar to a directory (also seen like that in the EA Project Browser). It may contain other packages or elements - in collections Packages and Elements respectively. Method EA.Repository.GetTreeSelectedPackage() gets the package selected in the EA Project Browser, either by right-clicking it and using Context Menu or by just being selected while using the main Menu.

EA.Element in the platform independent model represents each class. In order to get all elements selected in the EA Project Browser, one may use EA.Repository. GetTreeSelectedElements() method. Each class can have constraints on it and also can be in several (more specifically 0..n) associations. Constraints are stored in the Constraints collection whereas associations (that the element is part of) are in the Connectors collection.

EA.Constraint on an element (class) represents a constraint. A constraint may be of various types (attribute Type), one of which is OCL, and also has a name (attribute Name). The constraint itself is then stored in attribute Notes. The OCL constraint should be grammatically correct, because it is parsed by EA when stored.

EA.Connector (in the PIM, for the purposes of this work) is an association between two elements (classes) whose IDs it has stored as ClientID and SupplierID. In the newer version of EA connectors cannot have constraint of the OCL type and also there also is the unanswered question of the context of such constraint, and so they are not considered in this work.

2. Analysis

In short, the goal of this work is to create a prototype plugin for EA that is able to produce SQL code implementing constraints over a system specified by the selected PIM package that has these constraints defined in OCL.

In order to do so, there are several concerns that need to be addressed:

- EA Addin Workings
- OCL Constraint Extraction
- Getting DDL Information
- Parsing OCL
- Evaluating Parsed OCL
- Code Generation

As a part of analysis, several prototypes were made in order to find out what the given environment or tool is capable of. Not everything is documented and it would not be a good idea to just plan everything out of nothing without prior knowledge if the selected way is even possible. About EA and its concrete inner workings there will be made a special chapter, while in here only abstract things needed to be known before one starts to design a solution will be mentioned. By analysing the possibilities, slight fine tuning of the intended scope of this work should be made.

2.1 EA Addin Workings

In the prototype GetOCL it was ascertained that any .NET DLL implementing methods called for EA event handling (EA_GetMenuItems, EA_MenuClick, etc.) may serve as an EA Addin. EA uses COM for communication with its Addins and Windows registry for finding out about them - as has already been described in the Enterprise Architect chapter.

Also in the prototype GetDDL it was shown how to create GUI with Windows Forms and use it inside of an EA Addin. Finally by using an installer creator it is possible to make all the steps needed for the installation of an EA Addin by a simple installer that any user can use easily.

2.2 OCL Constraint Extraction

OCL constraints of the selected PIM package are located in its elements. While older versions of EA supported having OCL constraints located even on connectors, version 9 does not allow it. One may of course force it and manually enter the type of constraint on a connector as OCL, but EA does not check syntax, nor does it treat it like an OCL constraint. It has one very good reason and that is the intended context. By placing the constraint in the element, it automatically

gains the context of the class this element represents. Taking that into account it seems illogical to place them anywhere else and possibly cause confusion.

While the OCL language offers to express not only invariants, but also preconditions and postconditions, the usage of preconditions and postconditions is bound with operations, and as in this work no operations are considered, only OCL invariants are expected, the rest is ignored. The OCL invariants in EA look like this:

```
inv: constraint
```

Their context, as already mentioned, is determined by the element, and the name may be entered as a property which also serves for identification in EA. In the picture 2.1 from Example 5 is an OCL constraint of the class Flight with name hasPilot that is defined as:

```
inv: self.crew->includes(self.pilot)
```

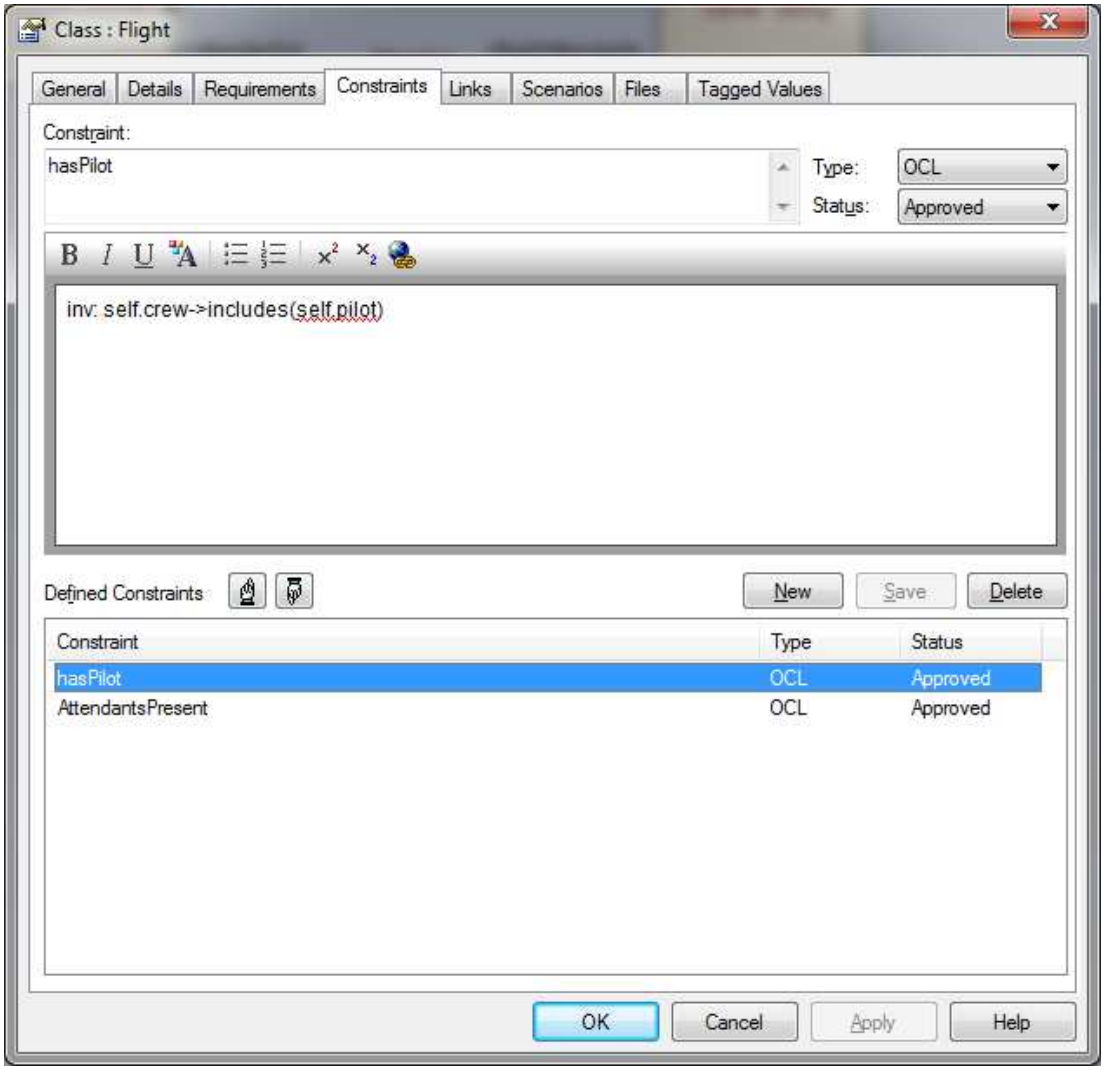


Figure 2.1: EA: element constraint tab

As proved by the prototype GetOCL, all constraints of the selected package may be easily obtained via the Automation Interface. Correctness of OCL syntax is checked by EA itself and so will not be a concern of this work.

2.3 Getting DDL Information

In order to implement the given constraints onto a database, the database schema has to be known. EA is able to transform the PIM package with constraints into PSM package with relational tables and then generate SQL DDL in the chosen dialect to create database according to the schema.

Due to the insufficient documentation and lack of examples of usage, a prototype had to be made to find how to access DDL package data and interpret it for the purpose of enhancing the already existing database. Prototype GetDDL shows how to look for an existing DDL package and that it is also possible to run the transformation and so create the package it through the Automation Interface. GetDDL takes the name of starting class and association role from the the PIM package through which one wants to navigate and it returns navigation data derived from the DDL package that can be used to generate part of SQL query that joins the needed tables from the database.

Because there are no direct links between the objects from the PIM package and the DDL package, another way of how to recognize what was the source of generation to the given object and vice versa has to be found. That may be accomplished by studying the EA built in transformation templates or by trying it on some examples, but preferably by both in order to be sure. A possible idea of how to make such mapping easier to recognize is by changing these transformation templates, however in that case if someone used the way they work for different purposes, then it might stop working. Therefore it might be better to just use the standard ones and let only those users who change the templates to worry of possible impacts such steps may cause.

PIM elements are mapped onto DDL tables (or more precisely elements of the table stereotype) with the same name. The PIM element's attributes are with the same name and the closest type possible used as the columns of the corresponding table. PIM associations however have several possibilities of being mapped depending on the cardinality of their ends. In the case of ono-to-one or one-to-many PIM associations, they are represented by one DDL connector of the same cardinality. Such a connector of course connects tables corresponding to the PIM elements between which lies the PIM connector. For simplicity let "one" in the cardinalities include also a zero cardinality as the behaviour is the same here.

On the other hand, many-to-many PIM connectors are mapped onto a special joining table and two one-to-many connectors that connect the joining table (cardinality is many on it's side) to the tables corresponding to the PIM elements between which lies the PIM connector (figure 2.3). The joining table however is not mapped onto any PIM element.

Advantage of such mapping is that, in the case of one-to-one and one-to-many DDL associations between the tables in the database, it may be enforced by the usage of foreign keys and IS NOT NULL checks, and also it eliminates possible redundancy which might have happened in the case of many-to-many associations. The connectors of the DDL package also contain information of how to join the two tables over the relation the connector represents. More specifically, the columns are in the name of the connector (column1 = column2) and the data in the relation may be gained by inner joining the tables on them.

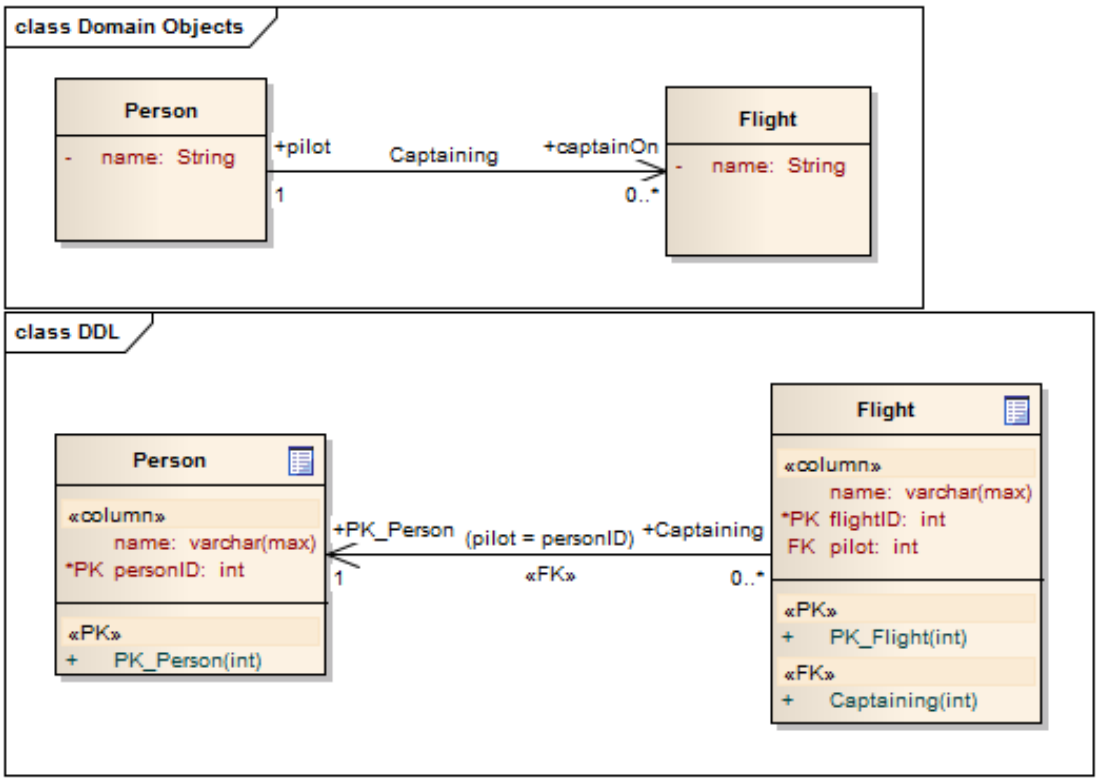


Figure 2.2: One to many asociation - PIM and DDL

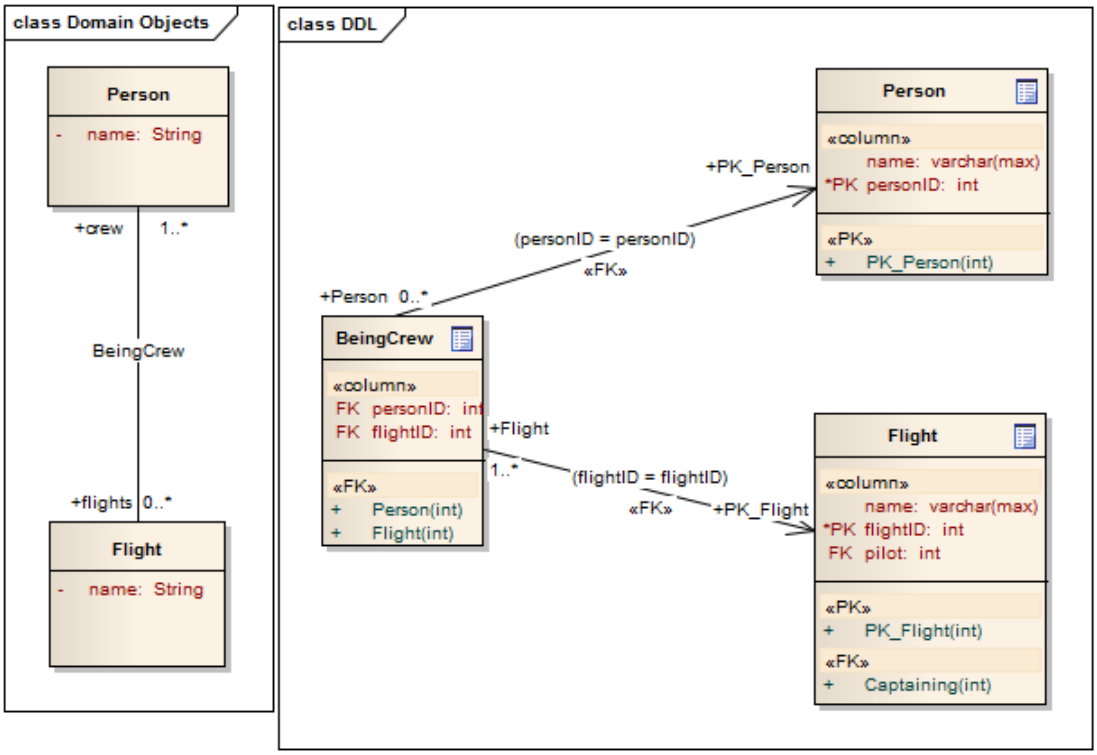


Figure 2.3: Many to many asociation - PIM and DDL

One of the encountered problems was, that when many-to-many associations from PIM package are unnamed, the joining table from the transformed DDL package is then always named JoinElement1ToElement2. But it is like that even

if there are more than one such connections between the same elements, and so the names of tables of course clash. However when the PIM connector has a name, the joining table of the many-to-many case is named after it and in the case of other cardinalities becomes the client role of the DDL connector. So by identifying the PIM connector through which one wants to navigate, they can easily find the corresponding structures in the DDL package for any standard case of cardinalities. Therefore, it is required for the PIM packages to have all their connectors named, so that the navigation may be unambiguous.

As an example of how to access DDL data for the purpose of transforming OCL into SQL can serve the PIM package from the figure 2.4 and OCL expression in the context of "Flight": "self.crew.name". At first in the targeted PIM package, the element with the name "Flight" has to be found, then, by going through all of his connectors, the PIM association with the role "crew" on the other side of the connector is located. After finding out whether that connector has many-to-many cardinality or not, the name of this PIM association "BeingCrew" is used to locate either the joining table or the connector in the DDL package. In this case the PIM connector is of the many-to-many cardinality and therefore DDL element with the name "BeingCrew" if found. To this element only two connectors are connected - one leading to the "Flight" table and the other to the target table "Person". In the names of these connectors are the columns needed for making a join between the two tables each connector connects. The first column is in both cases the one from the table "BeingCrew", because it is on the many side of both connectors that are of the one-to-many cardinality. Finally the "name" is a column in the target table ("Person"), and so, after using all the gathered information, the SQL query might look somewhat like this:

```
SELECT T3.name
FROM Flight T1
INNER JOIN BeingCrew T2 ON T1.flightID = T2.flightID
INNER JOIN Person T3 ON T2.personID = T3.personID
```

From the DDL diagram on the 2.5 picture it can be verified that it is indeed correct.

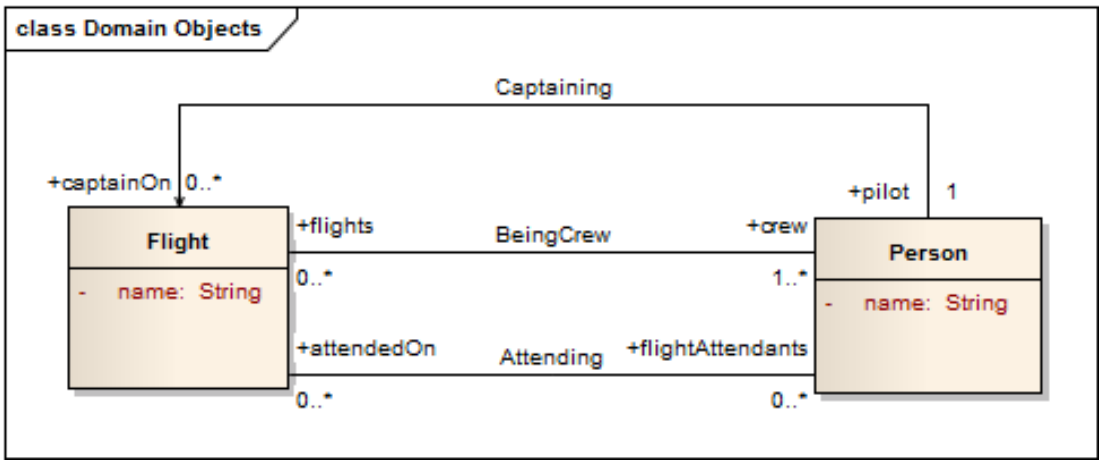


Figure 2.4: PIM package from Example 5

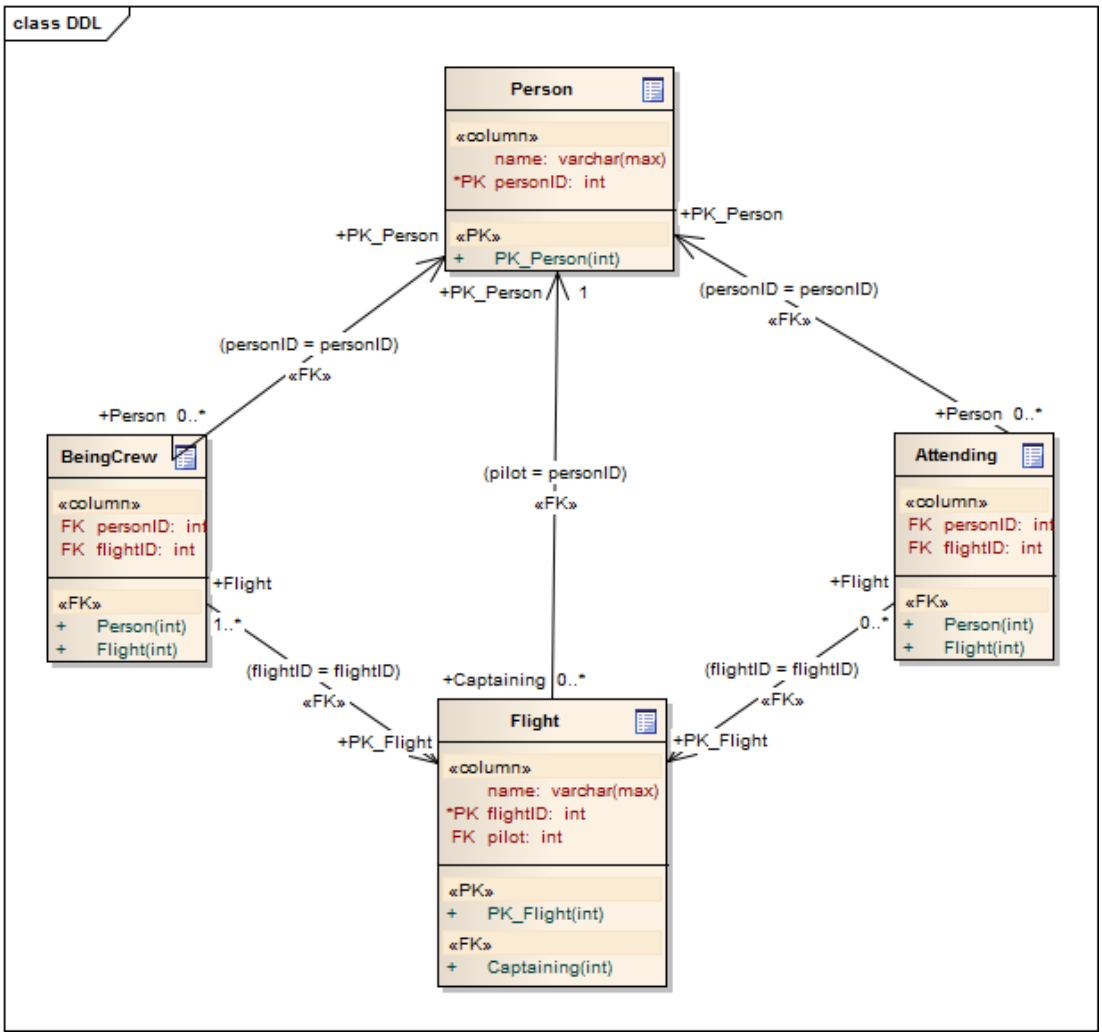


Figure 2.5: Transformed DDL package from Example 5

The GetDDL prototype tries to get the necessary data for joining the needed database tables for the given PIM element and the role through which to navigate by using the same algorithm as described. A simple Windows Forms window was created as its GUI for more comfortable usage.

As shown in the figure 2.6, when run on the same PIM package as the previous example for "Flight" as the starting point and "crew" as the wanted role to navigate over, it returns the FROM part of the SQL query that corresponds with the one derived manually from the example above.

2.4 Parsing OCL

In order to understand the available raw text of an OCL constraint, it has to be somehow parsed. While it might have been a good idea to use some third party parser to do it, despite having thoroughly searched on the Internet, no suitable .NET OCL2 parser has been found. Therefore it was decided to implement a new parser. This parser should be .NET based so that it is easily integrated into the .NET EA Addin. While it would be possible to make such a parser from scratch, it would not be easy, would take a lot of time and the probability of

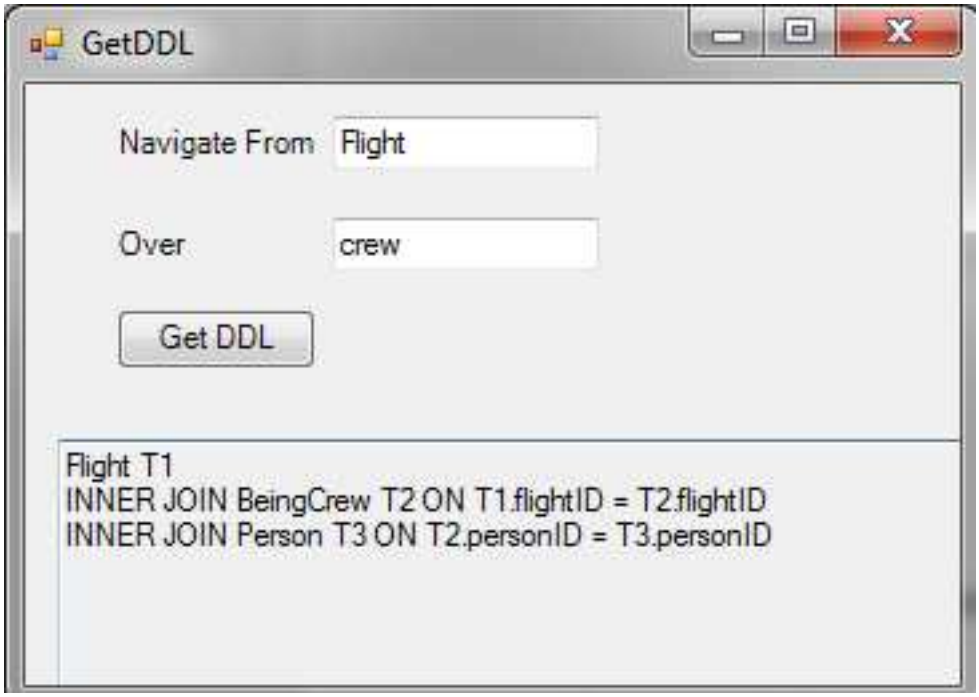


Figure 2.6: Prototype GetDDL: created window with output

mistakes would increase unnecessarily. And so it would be better to use some of the existing .NET parser generators.

There are many free parser generators available, such as ANTLR, GOLD Parsing System, etc. However ANTLR has a .NET version directly integrable into Visual Studio and its grammar may be kept separately from the handling code in the EBNF (Extended Backus-Naur Form) which enables easier and cleaner grammar development. Also on the Internet several OCL grammars in EBNF may be found for inspiration, albeit in various states of usability and completeness. But considering that OMG's standard is not available in a usable machine-readable form, any bit helps.

To ascertain usability of the chosen solution, prototype TryToParseOCL has been made where one can, by applying a chosen grammar, run through a directory on OCL constraints in files and see which are parsed correctly and which are not. The prototype also shows that the integration of ANTLR to the Visual Studio indeed works and that the generated code of the parser seems to be correct as well as easily used.

Because OCL grammar is quite extensive and covers not only invariants in the form of Boolean expressions (which is all that is needed for this work), it would be better to just use the part of grammar that recognizes exactly what this tool intents to transform. Also not every OCL Boolean invariants can be effectively transformed into SQL which means further simplifying of the grammar (and so of its handling code too) to the part that may produce more SQL-friendly code, seems to be quite a good idea.

Considering such a parser would take only a part of given OCL expressions that it is able to recognize and leave the rest be, it might be interesting to have the constraints using more obscure or SQL-friendly part of grammar to go through different parsers/evaluators that, while not producing as effective

code as the previous one, do their work. In effect, the tool then could have the constraints transformed into as effective code as possible, and at the same time achieve more completeness. Such parsers/evaluators connected similarly to the pattern of Pipes and Filters could have a queue of parsers that would parse the input OCL constraints and have their handling code to produce SQL code they are able to, while sending for them unrecognizable ones to the next parser. The last parser then may report the still not recognizable constraints to the user. The reason why this might be a good idea is that not every time one wants completeness for the price of producing practically unusable code that could slow their system unbearably, when it might be possible for the programmers themselves to implement the constraint another and, more importantly, usable way. The user could choose how far to go and what is to be left for him. Also implementations of grammars (and their handling code) just for the more direct and effective transformations would be a lot easier and transparent (and so less mistake-prone). As this work is just a prototype, only the first parser would be implemented and the idea itself would remain as a possible future extension.

2.5 Evaluating Parsed OCL

Each grammar rule may have a handling code which is called each time the rule is recognized, and so adds semantics to the syntax. As already mentioned, ANTLR uses partial classes in order to separate the rules and the implementation of their semantics - for any larger code one may use a method call in the handling code of the rule and the implementation of the method can be in the partial class of the parser, the stub of which is generated immediately upon adding the grammar file to the project. The principles were also tried in the TryToParseOCL prototype. The result of the evaluation should be the exact definition of the SQL queries to generate.

2.6 Code Generation

Given the definition of the SQL queries representing Boolean expressions, one should be able to generate SQL code from them. This could be done through a fixed interface so that the generation for different SQL dialects may be added later more easily.

However there still remains the question of how exactly to check for the constraint validation in a database. There are several possibilities (these may depend on what database it is targeted on):

- CHECK on a column
- Ttrigger on a table
- View for each constraint that returns violating data
- Periodical check of each constraint
- ...

In the case of using triggers it is also important to know which tables are the ones whose changes will cause the need for the validation, because the trigger needs to be assigned to each of these tables. Even worse it is for CHECK, because this construct needs to be applied to a specific column of a specific table, and so for more complex constraints each column somehow involved needs to have the CHECK on it with the same constraint validation.

On one hand, Periodical checks and aforementioned views do not have the problem of recognizing when to trigger them (first happens every preset period and the second is run by an external event), on the other hand, they may leave the database in an inconsistent state where the constraints are violated (in contrast with triggers and CHECKs). And so the user of the database (be it an application, integration script etc.) has to always be careful either of the time it uses the database (in the case of periodical checks) or to run appropriate views and see for himself if the constraints on the needed data are violated (in the case of views). Either way some sort of knowledge is needed in order to use the database which of course is not ideal and may cause a lot of problems.

An interesting option would be implementing a bit function for each constraint and have it used by the triggers. That way the code would be in one place which of course means less possibility to make mistakes, being more manageable and transparent. Also such function (if named accordingly to the constraints) may also serve the database developers for direct usage - for example they may serve in the case of batch data integration as a part of an import procedure while the normal triggers or checks are deactivated to increase performance.

2.7 Nonfunctional Requirements

As for the nonfunctional requirements, due to the fact that the expected amount of input OCL constraints as well as their size is low, there should be no real issues with speed or performance. However, because only a part of OCL grammar is targeted and just one SQL dialect is produced as a result, the future need to extend the scope of the application is evident. Therefore good extensibility should be aimed for. Also due to the amount of different tasks that need to be done for the plugin to function, some sort of reusability for its components might be appreciated by any future programmers needing to solve similar problems.

3. Transformations

In the analysis, it was already decided that only the part of the OCL 2 grammar that could be easily transformed into a usable SQL code, especially performance-wise, would be used. But what remains to be found is how to choose what part of grammar to use. Because the OCL 2 syntax is fairly extensive and because one of the main priorities is the effectiveness of the produced SQL, it seems to be a good idea to start from the SQL side of the problem. And so examining the most often used SQL constructs needed for implementing constraints and how to use them effectively should give good enough direction of where the transformations should be heading in order to be able to produce at least basic constraint generation. As for the SQL code efficiency, one of the more important goals is to reduce the number of used nested selects in comparison to larger selects that use joins by using the specified foreign keys between the tables. Not only that the nested queries typically make it harder for the DBMS Query Optimizer to optimally plan the queries, but also columns specified by the foreign keys are a logical place for indices to be placed. For example the query for finding what goods were sold on a given receipt may be:

```
SELECT G.Name
FROM Receipt R
    INNER JOIN ReceiptItem RI ON RI.ReceiptID = R.ID
    INNER JOIN Goods G ON G.ID = RI.GoodsID
WHERE R.Number = 1234

vs.

SELECT Name
FROM Goods
WHERE ID IN (
    SELECT GoodsID
    FROM ReceiptItem
    WHERE ReceiptID IN (
        SELECT ID
        FROM Receipt
        WHERE Number = 1234
    )
)
```

While this is a rather simple case, it is plain to see that the first query has the potential to not only be more efficient, but also the code is more readable and maintainable - it is similar to what a database developer would use in a hand-written procedure. Even if intelligent Query Optimizers may in some easy cases make an efficient plan for the second query too, it does not always have to be true and if the database is a larger one - for example several Terabytes of sale data, it may mean the difference between running several seconds as opposite to timing out.

While the plugin itself is counting on the possibility of adding other SQL dialect support, right now this work covers only T-SQL (runnable on Microsoft SQL Server 2008 R2), and so the structures described here will be in T-SQL.

Another important question that was already mentioned in the Analysis is how and when to check the constraint breach. SQL Server has triggers and so can do the checks automatically, it also offers views which can on the user's call show data that violate the constraint. However if a database user wanted to somehow check for violation in his own queries or procedures, triggers are practically useless and getting data from a view means to run a select on them and then using it as a nested query. More elegant way would be having the constraint check in a boolean function. Slight problem here is that in T-SQL there are no boolean functions, but rather bit functions - more specifically scalar functions that return type bit which is either 0 (false) or 1 (true). Their usage is similar to the boolean ones, just instead of directly using IF function() one has to use IF (function() = 1). The same way of usage holds also in the case of WHERE clause, for example WHERE (function() = 1). Each way of checking if the constraints holds has its advantages and disadvantages depending on the environment and the reason of their usage. And so the decision should be made by the users themselves who should have much better knowledge of the system the constraints are for.

3.1 Simple OCL Expressions

Presuming the SQL code for the boolean expression (that is false when the constraint is violated) is already generated, for simplicity let it be "1=1", the code for creating a bit function may look like this:

```
CREATE FUNCTION FunctionNameIsViolated()
RETURNS bit
AS
BEGIN
    DECLARE @result bit = (CASE WHEN 1=1
        THEN 0
        ELSE 1 END)
    RETURN @result
END
```

The name of each database object has to be unique and for the sake of users that could potentially use it should at least contain the name of the constraint it checks. And so the user should pick the names set in the name property in EA responsibly. When no such name is given, GUID is used to ensure the uniqueness of the name.The reason why it is better to have the inside Boolean expression be false when it is violated is that if there are more Boolean parts of the expression, the Boolean operators may stay the same, for example in the OCL invariant:

```
inv: A and B
```

Can then be symmetrically interpreted as the inside T-SQL expression:

```
(SQLexprA AND SQLexprB)
```

An exception is the OCL operator "implies" which does not exist in T-SQL. However it can be implemented by using AND and NOT operators, and so:

```
A implies B
```

May be interpreted as:

```
NOT (SQLexprA AND NOT(SQLexprB))
```

Arguably the most basic SQL construct is SELECT. Basic OCL expressions work at a time with single attributes - meaning the selects should too. The OCL entities and their attributes are directly mapped onto the tables and their columns in the database with the same name. and so for example an OCL expression:

```
Person.age
```

Would be mapped onto T-SQL expression:

```
SELECT age
FROM Person
```

OCL may also navigate over the defined associations of the PIM package. That, depending on the cardinality of the associations, corresponds to joining of one or two database tables. In the case of one-to-one or one-to-many associations one table and in the case of many-to-many associations two tables as described more thoroughly in the Analysis. For further examples let the PIM package in the figure 3.1 and its corresponding DDL package in figure 3.2 be used. If one wanted to get the age of all wives of all employees in OCL:

```
Company.employees.wife.age
```

But in SQL it would correspond to:

```
SELECT T4.age
FROM Company T1
    INNER JOIN Employment T2 ON T1.companyID = T2.companyID
    INNER JOIN Person T3 ON T2.personID = T3.personID
    INNER JOIN Person T4 ON T3.husband = T4.personID
```

How the data for joining are found is also in the Analysis. But the other thing to note is the aliases "T + number". Those are valid only in the scope of SELECT and any definition of the same alias in a nested query overrides the one above and so unless one wants to bind the nested query to the above one, they can generate the same aliases for each select and not care about the others.

Simple numbers and strings are the same in OCL as they are in T-SQL, they just have to be put under a SELECT clause. Arithmetic operations (including unary minus and NOT) are also the same, but it would not be very efficient to just mechanically put them between the already translated selects, not to mention wrong:

```
Person.age + 2 - 1
```

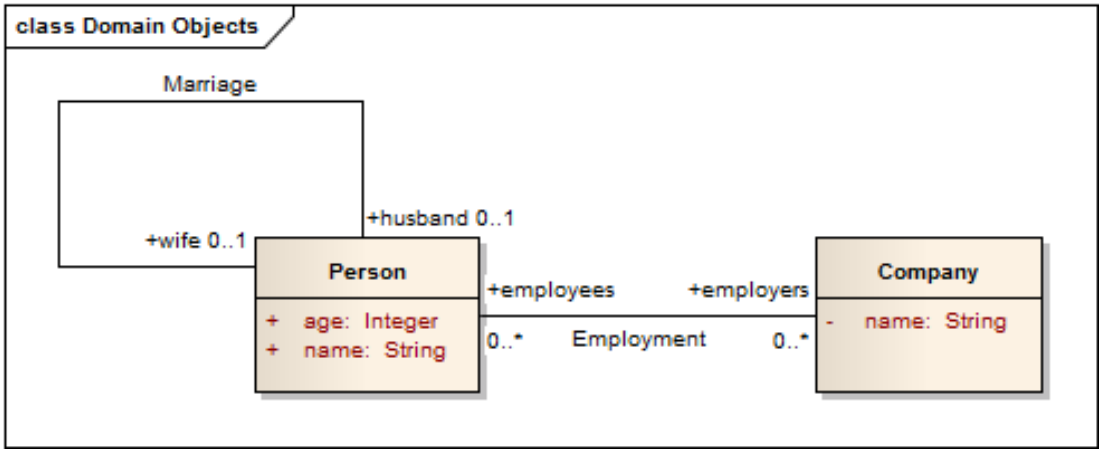


Figure 3.1: Example 4 PIM

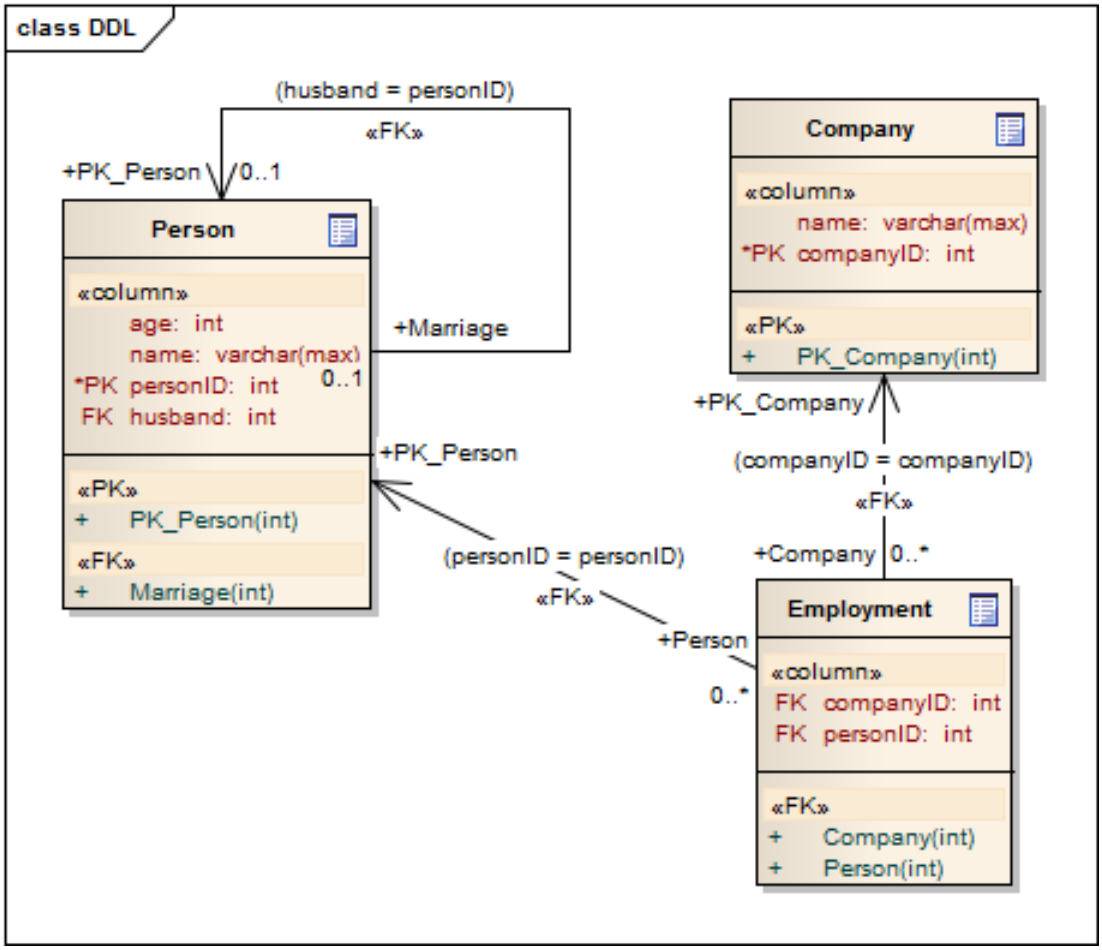


Figure 3.2: Example 4 DDL

Could be mechanically implemented in T-SQL as:

```
SELECT(
    SELECT (
        (SELECT age
        FROM Person)
        + (SELECT 2)
```

```
    )
    - (SELECT 1)
)
```

However the select that has some table in the FROM clause may return not only one value, but a whole set of them (this time the same as the count of rows in the table Person), whereas the other two selects return only one value, and so the query would fail if the row count was larger than 1. The OCL expression really means that for each row raise the age column by two and return one and return it as a set. While one may come with some elaborate ways of how to achieve that in T-SQL, probably the easiest and at the same time most eficient way would be to realize that some selects could be merged together into one select. Such merging can occur between selects where at most one of them has any tables in the FROM clause. The merge itself is pretty straightforward as it means that the FROM clause is filled with the FROM clause of the one that has something in it (or is left empty if both selects that are being merged have it empty) and the SELECT clauses are concatenated with the operator of the wanted operation between them. In the previous example that would mean

```
SELECT (age + 2 -1)
FROM Person
```

Huge advantage of this approach is that when parsing the grammar one could immediately make a select from a string or number without having to worry if on the way up the grammar tree the number or string is not used in an operation with a set of values like in the previous example instead of just with a single value. And also the efficiency is achieved in the single value case, because instead of:

```
SELECT ((SELECT 1) + (SELECT 2))
```

It would be:

```
SELECT (1 + 2)
```

But how about if for example a manager wanted to ensure that none of his companies employs people who have a wife that is not an adult? In OCL it is fairly straightforward:

```
Company.employees.wife.age >= 18
```

In T-SQL however there is a problem, because the type Boolean is not supported as a result of a select and so it cannot be implemented like:

```
SELECT T4.age >= 18
FROM Company T1
    INNER JOIN Employment T2 ON T1.companyID = T2.companyID
    INNER JOIN Person T3 ON T2.personID = T3.personID
    INNER JOIN Person T4 ON T3.husband = T4.personID
```

Not to mention that it would return a set as a result that would be as large as is the number of wives of the employees. What is really expected is for the constraint to hold only if all the employees' wives are adult. A trick that can be utilized is to, rather than be looking if all wives of the employees are adult, instead find out if there exist such employees' wives that are not adult:

```
NOT EXISTS
(
    SELECT 1
    FROM Company T1
        INNER JOIN Employment T2 ON T1.companyID = T2.companyID
        INNER JOIN Person T3 ON T2.personID = T3.personID
        INNER JOIN Person T4 ON T3.husband = T4.personID
    WHERE NOT ( T4.age >= 18)
)
```

And looking at the example above it is plain that with the trick above, all simple OCL boolean expressions can be implemented using T-SQL.

Put together with the function code the constraint named, for example, "areWivesOfEmployeesAdult" of the type "OCL" and slightly modified content to contain arithmetics too:

```
inv: Company.employees.wife.age >= 16 + 2
```

Could be implemented by a bit function like:

```
CREATE FUNCTION areWivesOfEmployeesAdultIsViolated()
RETURNS bit
AS
BEGIN
    DECLARE @result bit = (CASE WHEN
    NOT EXISTS
    (
        SELECT 1
        FROM Company T1
            INNER JOIN Employment T2 ON T1.companyID = T2.companyID
            INNER JOIN Person T3 ON T2.personID = T3.personID
            INNER JOIN Person T4 ON T3.husband = T4.personID
        WHERE NOT (T4.age >= 16 + 2)
    )
    THEN 0
    ELSE 1 END)
    RETURN @result
END
```

3.2 Self

Very problematic in OCL is the "self" operator. It binds all parts of constraint to the same row of the table that corresponds to the class used as a context and

that is repeated for each such row. For example a rule (in the context of Person) that says that only adults can have wives may also be written in this way:

```
(self.age > 18) implies (self.wife->notEmpty())
```

What it means is that when checking if it holds one has to go through every instance of Person (in T-SQL through every row of the table Person) and check the breach separately. It is plain to see that the use of "self" operator often forces less efficient code being produced, and so should be used only when necessary.

Whenever "self" is used in a constraint, whole constraint expression in T-SQL needs to be nested into a simple select that goes through each row of the context:

```
NOT EXISTS (SELECT tableID
    FROM ContextTable AS SELFT
    WHERE NOT (
        ...
    )
```

At the same time in every select that uses that table and its usage came out from translating the "self" operator, needs to be bound to that new outer select. More specifically, such nested selects needs to have in their WHERE clause:

```
T1.tableID = SELFT.tableID
```

Where T1 is the alias used for the first table in the FROM clause (it is there because of the "self" at the beginning of the OCL subexpression), "tableID" is the primary key of the table and "SELFT" is the alias of that new outer join that forces the check over every row of the context table.

As an easy example may serve the following OCL expression:

```
self.age >= 18
```

That can be interpreted in T-SQL as:

```
NOT EXISTS (SELECT PersonID
    FROM Person AS SELFT
    WHERE NOT (NOT EXISTS (SELECT 1
        FROM Person T1
        WHERE (T1.PersonID = SELFT.PersonID) AND NOT (age >= 18))
    )

)
```

Going for a more difficult and complete example where the usage of "self" might be justified by using an OCL constraint saying that everyone should be married:

```
inv: self.wife->notEmpty() or self.husband->notEmpty()
```

Produces the following code for the creation of a bit function that returns 1 if the constraint is violated and 0 if it holds:

```
CREATE FUNCTION [EveryoneIsMarriedIsViolated]()
RETURNS bit
AS
BEGIN
    DECLARE @result bit = (CASE WHEN
    NOT EXISTS (SELECT PersonID
        FROM Person AS SELFT
        WHERE NOT (((SELECT COUNT(*)
            FROM Person T1
            INNER JOIN Person T2 ON T1.husband = T2.personID
            WHERE (T1.PersonID = SELFT.PersonID)) > 0)
    OR ((SELECT COUNT(*)
        FROM Person T1
        WHERE (T1.PersonID = SELFT.PersonID)) > 0)))
    THEN 0
    ELSE 1 END)

    RETURN @result
END
```

3.3 OCL Functions

And finally also some of the OCL functions should be implemented. The meaning of the functions will be defined and the way the functions are implemented will just be shown as an example of the mapping from OCL into SQL with some comment where necessary. The list of the implemented functions will of course grow in time, but for the purpose of the thesis only a selected set from the OCL standard library will be implemented as an example.

3.3.1 size()

Returns the size of a collection. For example the number of people in the system:

```
Person->size()
```

May be interpreted as:

```
SELECT COUNT(*)
FROM Person
```

3.3.2 sum()

Sums the values of the attribute it was called on over the collection. For example making a sum of the weight of all boxes:

```
Box.weight->sum()
```

May be written as:

```
SELECT SUM(weight)
FROM Box
```

3.3.3 max()

Returns the maximum of the attribute values it was called on over the collection. For example finding out how old is the oldest person in the system:

```
Person.age->max()
```

Can be:

```
SELECT MAX(age)
FROM Person
```

3.3.4 min()

Returns the minimum of the attribute values it was called on over the collection. For example finding out the smallest person:

```
Person.height->min()
```

May be:

```
SELECT MIN(height)
FROM Person
```

3.3.5 average()

Returns the average of the attribute values it was called on over the collection. For example finding out the average wage of all employees:

```
Company.employees.wage->average()
```

In T-SQL:

```
SELECT AVG(wage)
FROM Company T1
INNER JOIN Employee T2 ON T1.CompanyID = T2.Employed
```

3.3.6 isEmpty()

Returns true if the collection over which it was called is empty, false otherwise. For example every man in the system for finding a date for men has to be single:

```
Man.wife->isEmpty()
```

Which can be written as:

```
(SELECT COUNT(*)
FROM Man T1
INNER JOIN Woman T2 ON T1.married = T2.WomanID
) = 0
```

3.3.7 notEmpty()

Returns true if the collection over which it was called is not empty, false otherwise. A more complex example (everyone has to be married) was already used when describing the Self operator.

3.3.8 includes(Collection)

Returns true if the collection it was called on includes any of the elements in the given second collection, false otherwise. For example a pilot has to be on board of the plane (Example 5):

```
self.crew->includes(self.pilot)
```

Which may be interpreted as:

```
NOT EXISTS (SELECT FlightID
    FROM Flight AS SELFT
    WHERE NOT (EXISTS(((SELECT flightID
        FROM Flight T1
        INNER JOIN BeingCrew T2 ON T1.flightID = T2.flightID
        INNER JOIN Person T3 ON T2.personID = T3.personID
        WHERE (T1.FlightID = SELFT.FlightID)))
        INTERSECT
        ((SELECT pilot
        FROM Flight T1
        WHERE (T1.FlightID = SELFT.FlightID))))
    )
)
```

3.3.9 includesAll(Collection)

Returns true if the collection it was called on includes any of the elements in the given second collection, false otherwise. For example all attendants have to be on board of the plane (Example 5):

```
inv: self.crew->includesAll(self.flightAttendants)
```

Which in T-SQL:

```
NOT EXISTS (SELECT FlightID
    FROM Flight AS SELFT
    WHERE NOT (NOT EXISTS(((SELECT flightID
        FROM Flight T1
        INNER JOIN BeingCrew T2 ON T1.flightID = T2.flightID
        INNER JOIN Person T3 ON T2.personID = T3.personID
        WHERE (T1.FlightID = SELFT.FlightID)))
        EXCEPT
        ((SELECT flightID
        FROM Flight T1
```

```
        INNER JOIN Attending T2 ON T1.flightID = T2.flightID
        INNER JOIN Person T3 ON T2.personID = T3.personID
        WHERE (T1.FlightID = SELFT.FlightID))))
    )
)
```

4. Project Documentation

OCLtoSQL is a plugin of Enterprise Architect (EA) that is able to extract OCL constraints out of a selected PIM package, and by using data from corresponding DDL package, produces SQL code implementing them over the database both packages represent.

4.1 Configuration Management

4.1.1 Choice of Tools and Technologies

The language chosen for the implementation of the OCLtoSQL plugin was C#, because EA Automation Interface supports .NET based languages and C# is a simple modern and object-oriented language with large enough support base to easily find how to use it in many typical situations which of course may save the programmer a lot of trouble and time.

As for IDE, the choice was pretty obvious - Visual Studio 2010, arguably one of if not the best IDEs available that has full C# and many other technologies support, and is for academical usage free of charge through MSDN Academic Alliance ([14]). Similarly Microsoft SQL Server can be freely used to test the produced SQL code.

In order to store and develop code safely, some type of source control should be used. SVN is a simple, yet fairly often employed, variant that has already many times been proven to be reliable and functionality-wise advanced enough. Ankh ([1]) is an extension of VS 2010 that integrates SVN directly into the IDE and makes SVN usage nearly effortless.

For the reasons already described in the Analysis (C#, integrable into VS, large support base, existing partly usable OCL grammars), ANTLR ([2]) was chosen as the parser generator used in the OCL code translation. The integration into the Visual Studio 2010 is done by extensions Tunnel Vision Labs Extensibility Framework, ANTLR Language Support and String Template 4 Language Support, all found via Online Gallery in Visual Studio Extension Manager.

One of the easier to use tools for documentation generating is Doxygen ([8]) which can scan the "///" comments, the structure of which is automatically made by VS and generate a simple documentation out of it, describing the public interface of each component. The configuration is made easy by Doxygen Wizard that can produce the Doxygen configuration file and run the generation by itself from a GUI.

And of course the thesis itself should be made into a pdf that satisfies all the faculty"s standards. Most of them are automatically forced by using the official Latex template found at [11]. As a setting tool MikTex ([13]) is sufficient and is targeted on Windows. The pdf compilation itself is done by using its tools cslatex, dvips and ps2pdf - ideally run by one script to make it efficient. One of the inconvenience of this is that only figures in EPS format may be included, however a simple yet useful tool for batch PNG to EPS conversion (found at [18]) is able to solve a problem in a few minutes.

4.1.2 Hosting and Backup

Assembla.com is a website that offers free project hosting. Each project can have its own SVN repository (but also Git is supported), Wiki, Ticket Management System, ability to authorize level of access, and a lot of other functionality. However the most important is the SVN repository that is safely stored on the internet which along with the local copy decreases the chance of losing the code in the case that something happens.

Excellent for document creation and storage is GoogleDoc which is one of the free services available under a Google account. In there one may not only read and edit documents from any place with internet access, but also share them with others (in example the Advisor of their work) and let them comment what should be written differently, what is missing etc. Such comments in a document may then be discussed and then marked as solved, all the while all involved being informed of it via email. Included are of course English spell-checking and correct word suggestions. Again, for the purpose of decreasing the chance of losing data, each time after making more changes, a copy of the document may be exported (for example in the Rich Text Format) and stored locally.

The SVN repository contains:

Prototypes Source code for the prototypes:

- GetDDL
- GetOCL
- TryToParseOCL

Trunk Source code of the OCLtoSQL plugin.

Master Thesis Source code and images needed for the compilation of this thesis into PDF.

4.1.3 Deploy

The installation of an EA plugin needs to copy somewhere the needed DLL(s), register the main DDL for COM interoperability and writing into the registry of EA to let it know the plugin exists. To want from the user to do it all manually would be inconvenient. Not to mention when wanting to uninstall it some time later, one could hardly remember all that was needed to be removed. In this case an installer that could automate each of these steps would be great. WiX ([23]) is a tool that creates a MSI package according to the provided XML definitions. MSI package is a file that is able to specify the steps above (and many more things) for the Windows Installer to do and later to undo them (for example from the Programs control in the system). WiX project can be added into the Visual Studio solution (and set to depend on the main project of the EA plugin) and produce the MSI file automatically on building the solution.

4.1.4 Target Platform

OCLtoSQL is primarily targeted on Enterprise Architect 8.0 (as that is the version the faculty has licence for) running on Windows 7 with .NET Framework 4. It is

a prototype and so that is considered enough to guarantee. However, it should also work on newer versions of Enterprise Architect than 8.0 and Windows from XP up (as long as the .Net Framework 4 is installed).

4.1.5 Tools Overview and Versions

- Visual Studio 2010 Ultimate
- SQL Server 2008 R2 Developer
- Ankh 2.3 (VS 2010 Extension)
- Tunnel Vision Labs Extensibility Framework 1.0.7 (VS 2010 Extension)
- ANTLR Language Support 1.0.6 (VS 2010 Extension)
- String Template 4 Language Support 1.0.6 (VS 2010 Extension)
- ANTLR .NET C#3 Bootstrap 3.3.1 (directly copied in the project)
- WiX 3.5
- Doxygen 1.8
- MikTeX 2.9

4.2 Architecture

4.2.1 Logical View

OCLtoSQL may be divided into 6 main parts (figure 4.1):

GetOCL Gets all OCL constraints out of the selected PIM package in EA. Returns name, context and the code for each constraint.

DDLPackageControl Is able to create and validate a DDL package under the selected PIM package.

DDLPackage Gets the data needed for joining tables in the database that correspond to the navigation in the PIM package.

OCLtoSQLTranslator Translates the given OCL constraint into basic SQL structures that implement it.

TSQLGenerator Generates T-SQL code out of the given basic SQL structures.

OCLtoSQL Is the EA plugin itself that contains a simple Gui and uses the other components to achieve the wanted functionality.

Sequence Diagram of what happens on initiating SQL code generation from a PIM package is shown in the figure 4.2.

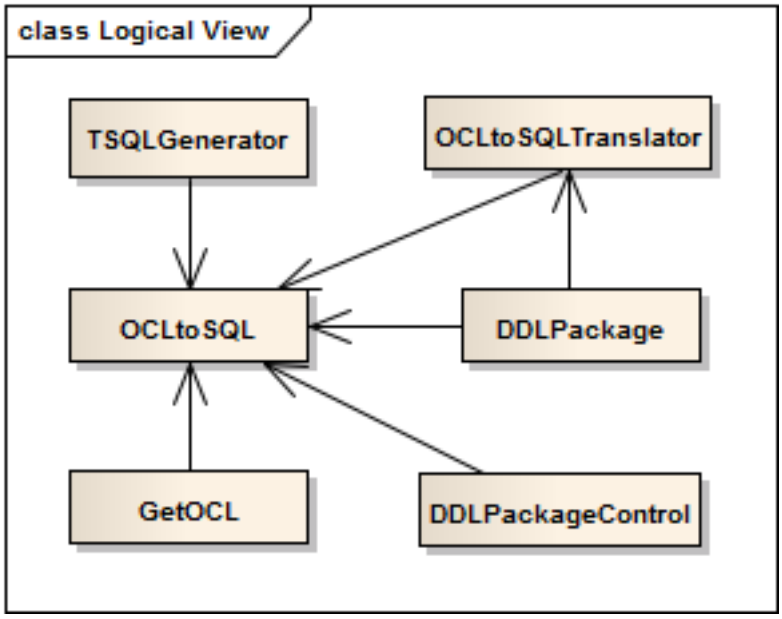


Figure 4.1: Logical View of OCLtoSQL

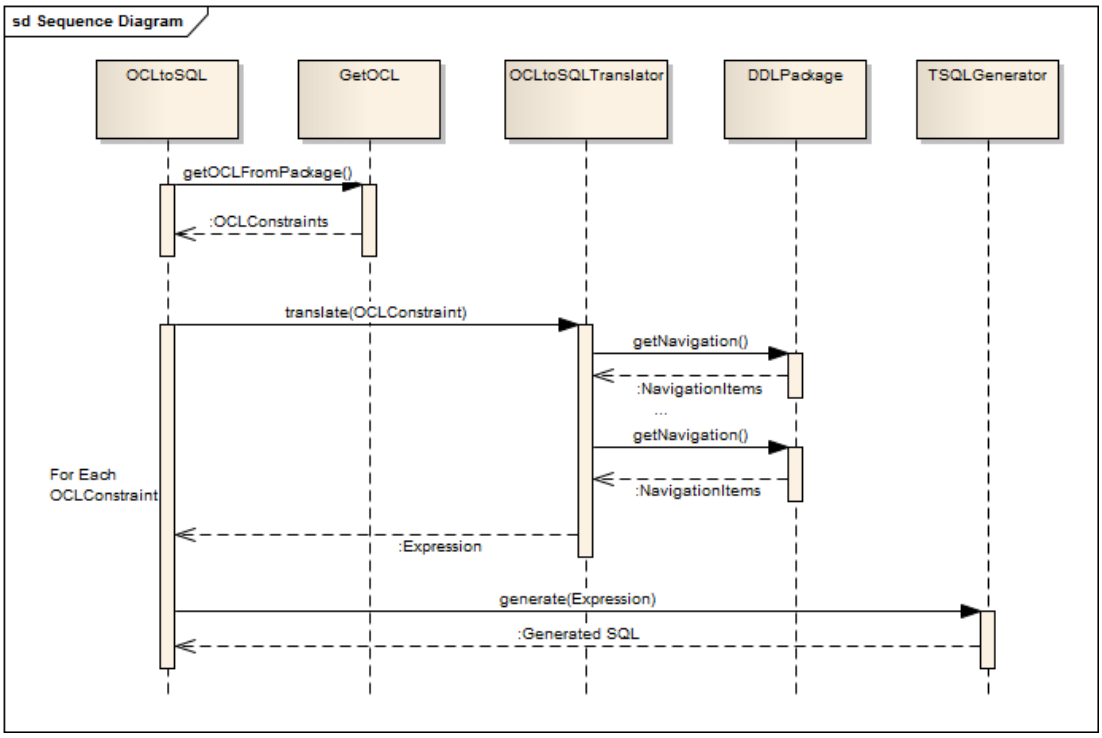


Figure 4.2: Sequence Diagram of the SQL generation from OCL in OCLtoSQL

4.2.2 Development View

OCLtoSQL consists of 6 separate DLL projects and 1 WiX project, placed in the same solution, that are connected via Project References. Advantage of Project References is of course the fact that when building a DLL, all DLLs that the one being built depends on are automatically built too. And of course developing itself is more comfortable, because the programmer does not need to have open 7 VS windows (one for each DLL project) instead of just one.

Interesting to note is that the separate DLLs may be later reused more easily than if all was just in one assembly, and also it is easier to replace one of them with a different one (from example OCLtoSQLTranslator with different SQL dialect).

Contained projects (figure 4.3):

GetOCL Gets all OCL constraints out of the selected PIM package in EA.

DDLPackage EA DDL package access and control.

OCLtoSQLTranslator Translation of OCL to common SQL structures.

TSQLGenerator Generates T-SQL code.

OCLtoSQL Plugin, GUI and runs it all.

OCLtoSQLSetup WiX project that builds the installer. Because it depends on the main DLL (OCLtoSQL) which depends on the rest of projects, if there is a change anywhere in the solution the installer is automatically rebuilt too.

CommonSQL Contains basic SQL and Expression objects used by several projects.

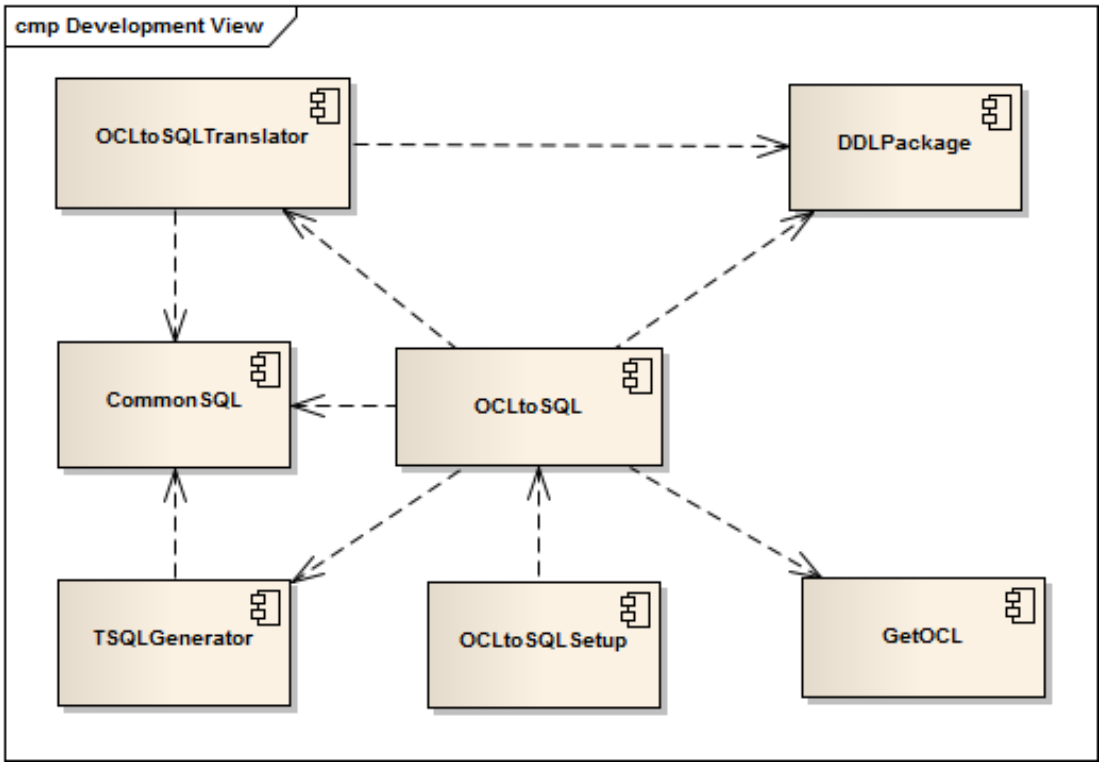


Figure 4.3: Component diagram of OCLtoSQL

OCLtoSQL also utilizes (figure 4.4):

Windows Forms Gets all OCL constraints out of the selected PIM package in EA.

Interop.EA EA DDL package access and control.

Antlr3.Runtime Translation of OCL to common SQL structures.

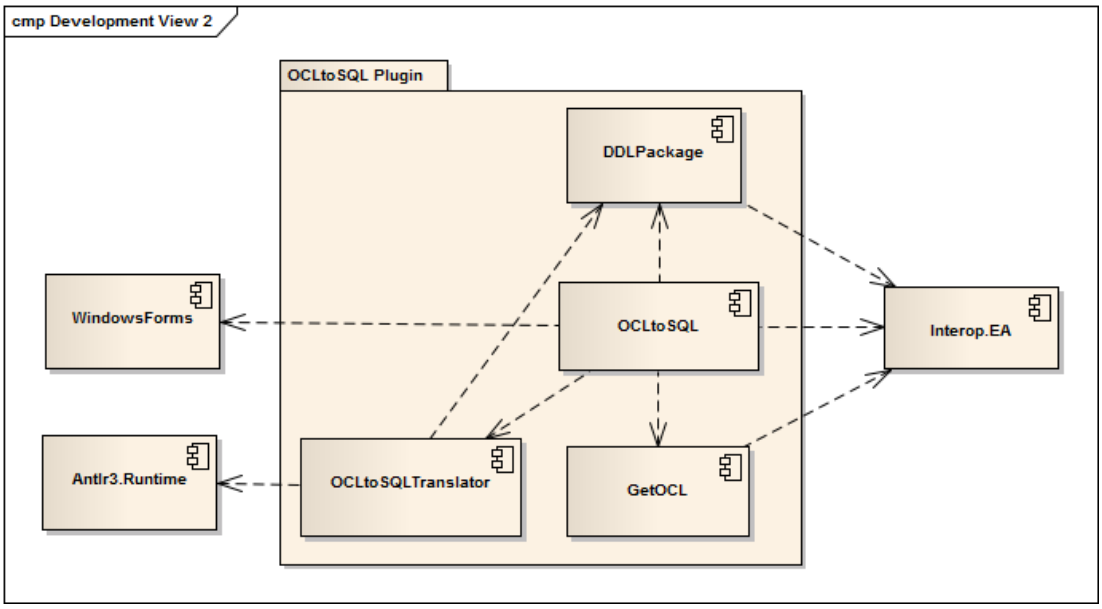


Figure 4.4: Outer component utilization

4.3 Programmer Guide

Rather than a detailed documentation which is already available in a separate generated reference documentation, this part discusses some of the less trivial usages of external tools.

4.3.1 ANTLR

ANTLR is a very powerful if, especially in the beginning, very hard to master. There of course exist many guides, but out of them [3], [4] and [5] seem to be probably the most useful to at least start learning how to use ANTLR. Integration to VS 2010 is described in [17]. As the main starting point of the grammar creation served the OCL grammar from [15] which was later greatly modified. One of the more advanced features that was used inside, the ANTLR predicates, are introduced in [6].

4.3.2 WiX

How to add a WiX project into VS is described more than sufficiently here [22]. Extremely useful proved [21] which describes a very similar situation to the one in this work, however it uses Sharp Developer instead of Visual Studio 2010, and so the process needs to be modified. Probably the easiest way is to leave everything in one .wsx file. At least two components are necessary in the installer, because registry for all users and one user cannot be modified in the same component. The components may be placed directly in the Directory mapping and their name has to be referenced in the feature list. The files, represented by the "File" element need to be taken from the Release directory and consist of all .dll files needed for the plugin to run. Registry changes are in the "RegistryValue" elements - the one adding the plugin name and its main class name into EA registry can be easily hand-made, but the registry entries that register the main plugin DLL for

COM interoperability have to be generated. The usage of the Heat tool and the renaming of the result parts can stay the same as described in [21], however then the registry entries can be copied to one of the components placed in the lone .wxs file as already mentioned.

Important is to note that none of the GUIDs should be copied from the guides, but rather they should be regenerated, for example by using the inbuilt GUID generation tool in VS 2010, the ones generated by the Heat tool are of course already unique and can stay as they are.

One of the really nice features of WiX is that it can produce a standardized GUI for the installer just by adding a reference to WixUIExtension.dll (located in the WiX installation) and one line in the .wxs file that selects which ine is to be used (of course the more sophisticated ones need further information, but that is not necessary in this case):

```
<UIRef Id="WixUI_Minimal"/>
```

4.4 User Guide

User guide contains useful information about the requirements and usage of OCLtoSQL EA plugin and leads through the process of installation and uninstallation. More up-to-date information may be found at [16] where is the OCLtoSQL project hosted.

4.4.1 Requirements

OCLtoSQL is primarily targeted and tested on Enterprise Architect 8.0 running on Windows 7 with .NET Framework 4 installed. Other versions of EA as well as Windows (with .NET Framework 4) can work too, but are not guaranteed. The plugin takes less than 1MB of disk space in Program Files.

4.4.2 Installation

Installation of the OCLtoSQL plugin is pretty straightforward, bucause all that is needed is to run the OCLtoSQL.msi file that does all the work. OCLtoSQL uses Common Public Licence, and so during the installation the user has to agree to the licence terms (figure 4.5). Right after the installation, whenever Enterprise Architect is run, it should have OCLtoSQL available in its addins (figure 4.6).

4.4.3 Usage

Apart from the Settings and About, all OCLtoSQL plugin operations have to be run on a PIM package that has its classes as children in the repository hierarchy (figure 4.7). Each association in this PIM package should be named (serves as an identification in the DDL package) as well as its roles (used by OCL for navigation).

As seen in figure 4.8, OCLtoSQL offers the following operations:

Settings Configuration of the SQL code generation.

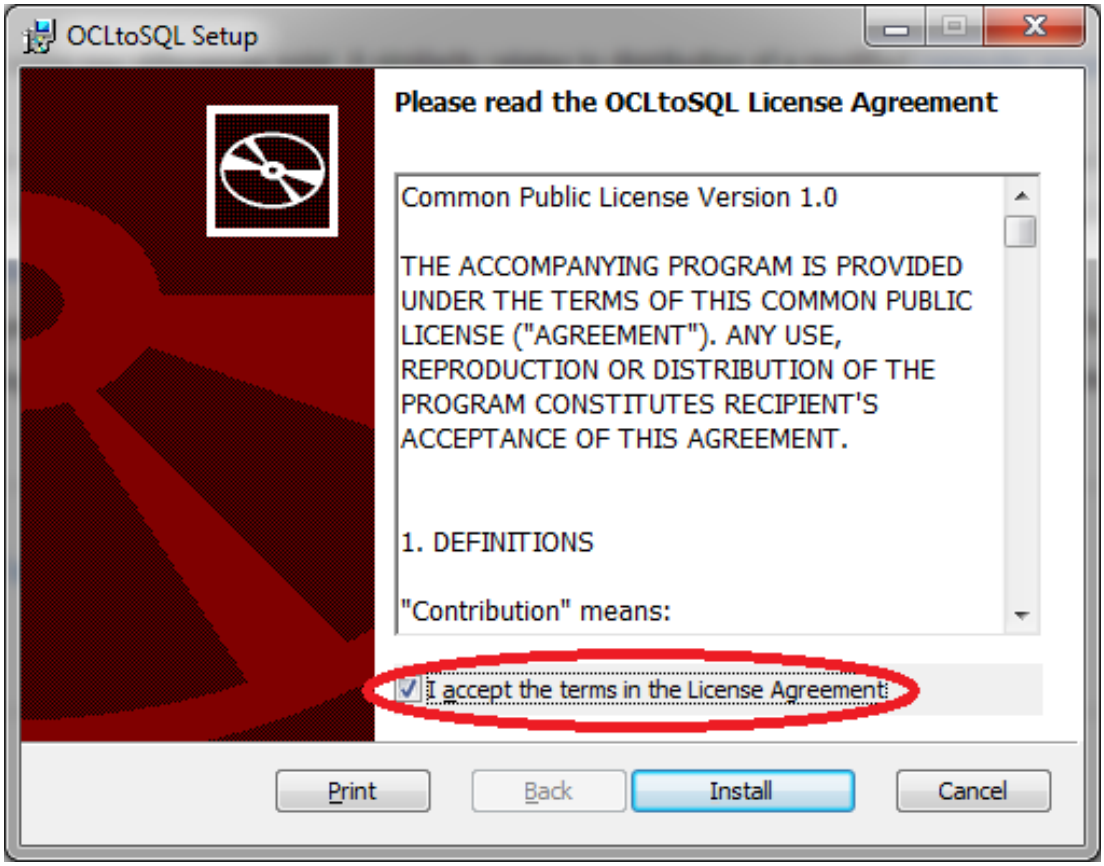


Figure 4.5: Installation of OCLtoSQL: accepting the Licence Terms

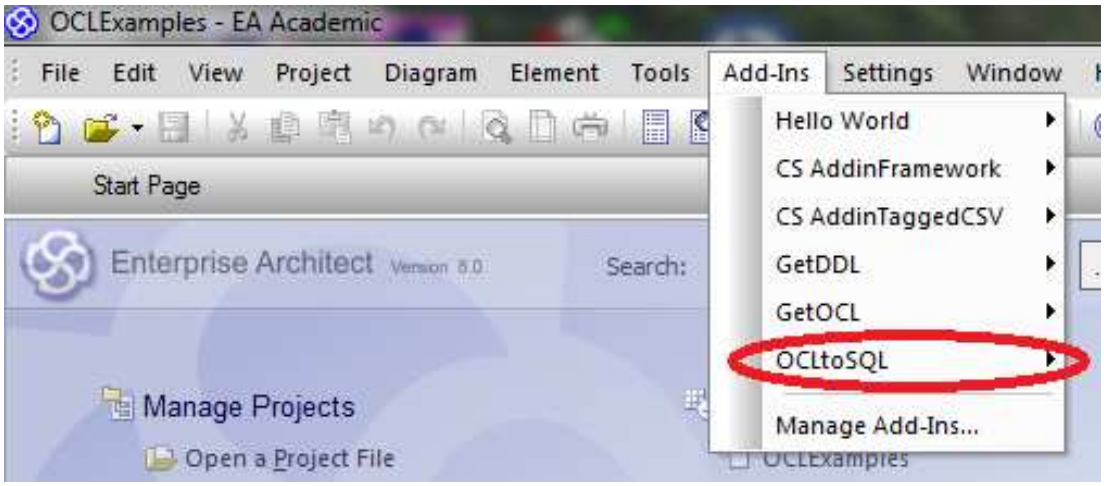


Figure 4.6: After the installation: addin available in EA

Create DDL Creates a DDL package directly under the selected PIM package by using the standard EA transformation. The new DDL package contains information about the database on which the constraints should be implemented. EA can also use it to generate SQL code that creates such database.

Evaluate DDL Ensures that the created DDL package under the selected PIM package has all its data correctly updated. EA sometimes unfortunately in

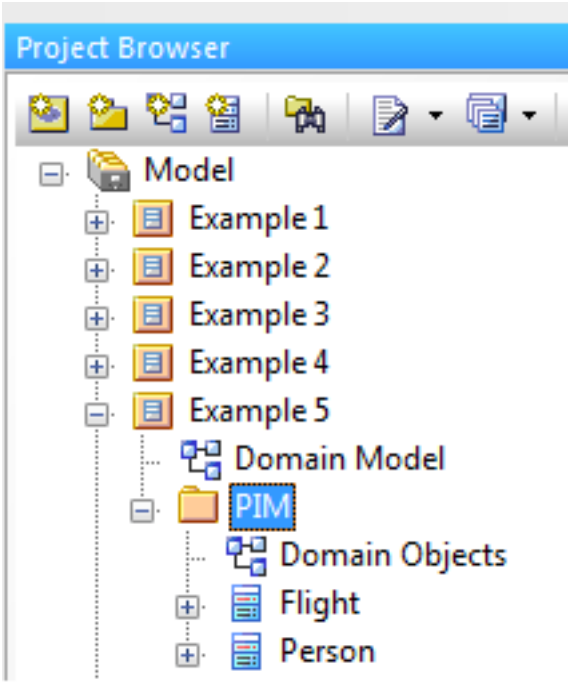


Figure 4.7: OCLtoSQL should be called on a PIM package

the transformation to DDL leaves some of the data out and so this may be necessary.

SQL Generation After the DDL package under the selected PIM package is properly created and evaluated, this option runs the SQL code generation of all OCL constraints in the elements of the PIM package and returns it as a whole runnable script as shown in figure 4.9. The result script can then be saved as a file or copied to clipboard by using the available buttons "Save As" and "Copy" respectively.

About Basic information about the plugin.

The produced result code (figure 4.9) has a header (hidden in a comment) that informs about it having been generated by OCLtoSQL and the time it has happened as well as how many OCL constraints were successfully parsed. Then follows for each constraint either an error or the correct SQL code that implements it in the database. The errors serve as a warning for the developer that he either has to write the constraints differently or has to implement them in SQL code on his own.

4.4.4 Uninstallation

Uninstalling OCLtoSQL is possible by the standard way in the Windows (by going to "Control Panel/Programs/Uninstall a program", right-clicking on OCLtoSQL in the list and clicking "Uninstall" in the context menu). Because the OCLtoSQL plugin uses the standard Windows Installer for installation, the uninstallation always removes all the changes made in the system (both in Program Files as well as Registry).

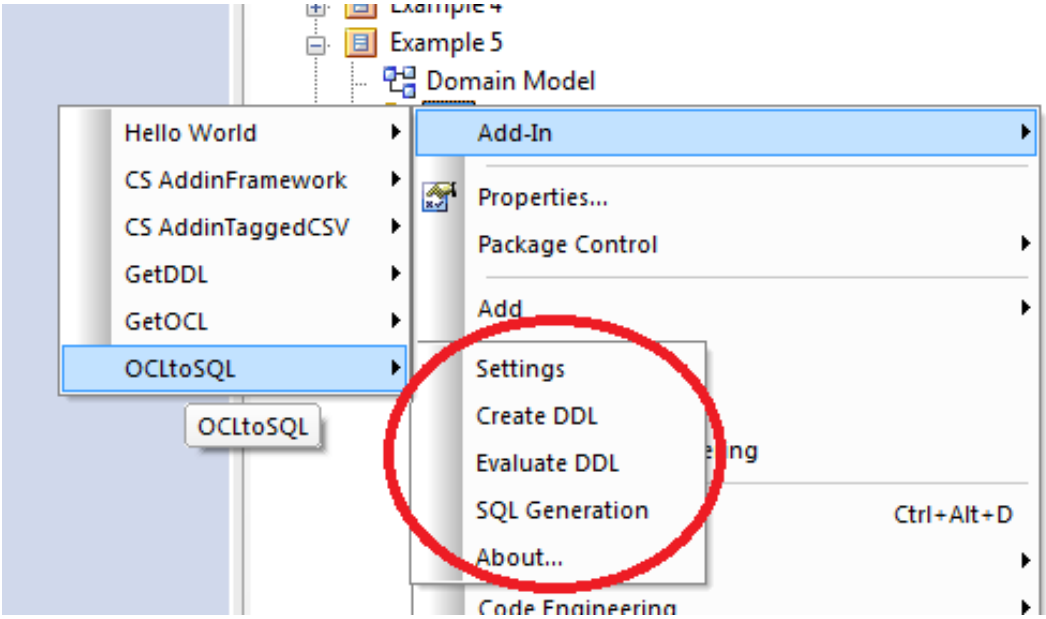


Figure 4.8: Operations offered by OCLtoSQL

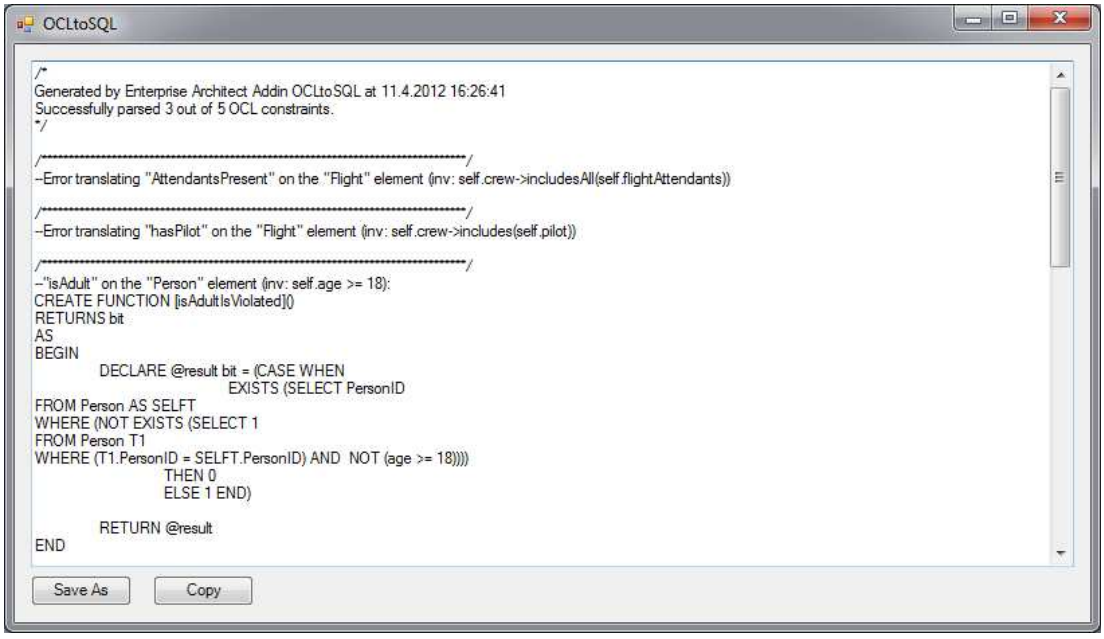


Figure 4.9: OCLtoSQL result window

Conclusions

The result of this work is Enterprise Architect plugin OCLtoSQL that is able to take OCL constraints directly from the selected PIM package and produce corresponding T-SQL code that could implement them in the database where the entities from the PIM package live. The architecture of the plugin, in which the SQL code generation is placed into a separate DLL, favours the possible future expansion by different SQL dialects. Having the part of OCL grammar that is used for parsing the OCL constraints in EBNF form means better readability and maintainability as well as making it easier to add more transformations. Separation of the DLL projects for getting OCL constraints or communicating with DDL package from EA as well as the one doing the transformations may also contribute to the possible future reusability of the code elsewhere.

The assignment of this work was not easy. It required knowledge both in the Software Engineering (concepts of MDA, machine-readable specification, OCL...) and the Database Systems (SQL, DBMS, MS SQL Server...). Also needed was exploring the Enterprise Architect environment despite not very beginner-friendly documentation. Some of the used tools, like for example ANTLR, despite being useful, have a relatively steep learning curve. Put together it means that quite a lot of research was required before the actual implementation of the plugin could occur.

As has already been discussed, there are some more complete implementations of OCL to SQL transformations, albeit usually on Java platform, whereas this plugin exists purely on .NET platform and is directly usable in Enterprise Architect. However this work offers a somewhat different approach by right from the parsing itself being closer to the SQL and trying to map the OCL into SQL in a simple and direct way, rather than mapping it on some OCL common library which is better suited for more imperative languages, like C# or Java. OCL is a query based language like SQL and so similar in a lot of ways. The more direct way of the transformation enables to better optimize the result SQL queries and to create them looking closer to the way they would look if they were implemented by hand. Also interesting proved to be the idea of analysing the needed transformations from the SQL side as that is the side of which a good working code is expected, and later adding more OCL functionality while trying to keep the produced code efficient.

There are of course a lot of parts in this work that might be improved. The first area that comes to mind is making the OCL to SQL transformation more complete (by covering more of OCL). By using the ANTLR error recovery ability the plugin could, when an error occurs, show suggestions of what is wrong. Already hinted was having more SQL dialects generation available. Further optimization of the produced SQL code is always a possibility. Nice would also be having more kinds of output available (not only functions, but also triggers, views, different naming conventions...). Anyway, it is plain to see that there is a lot of potential for future enhancement as the prototype plugin itself was designed taking that into account.

All in all, it can be said that OCLtoSQL plugin has met all the assigned requirements and even though there is still a lot of areas where it could do better (as there is in practically any software ever made), it is a useful tool that might

find some application not only in the academic world, but also in practice. Hopefully, some of the ideas and mistakes shown in this work will help in some possible future more successful endeavours in the area of MDA and the relation of OCL to SQL.

Bibliography

[1] *Ankh SVN - subversion support for Visual Studio.* [10-04-2012] http://ankhsvn.open.collab.net/

[2] *ANTLR - ANother Tool for Language Recognition.* [10-04-2012] http://www.antlr.org/

[3] *ANTLR 5 min introduction.* [10-04-2012] http://www.antlr.org/wiki/display/ANTLR3/Five+minute+introduction+to+ANTLR+3

[4] *ANTLR example.* [10-04-2012] http://www.codeproject.com/Articles/28294/a-Tiny-Parser-Generator-v1-2

[5] *ANTLR3 Parser Grammars.* [10-04-2012] http://www.antlr.org/wiki/display/ANTLR3/Quick+Starter+on+Parser+Grammars+-+No+Past+Experience+Required

[6] *ANTLR Predicates.* [10-04-2012] https://wincent.com/wiki/ANTLR_predicates

[7] *Constraint Support in MDA tools: a Survey.* [09-02-2012] http://jordicabot.com/research/OCLSurvey/

[8] *Doxygen.* [10-04-2012] http://www.stack.nl/~dimitri/doxygen/index.html

[9] *Dresden OCL.* [01-02-2012] http://www.dresden-ocl.org

[10] *Enterprise Architect User Guide.* Sparx Systems, 2005.

[11] *Faculty of Mathematics and Physics Latex Master Thesis template .* [03-01-2012] http://www.mff.cuni.cz/win.cs/studium/bcmgr/prace/dp_sablona.htm

[12] *How to use OCL22SQL.* [01-02-2012] http://dresden-ocl.sourceforge.net/usage/ocl22sql/index.html

[13] *MikTex - TeX implementation for Windows.* [10-04-2012] http://miktex.org

[14] *MSDN Academic Alliance.* [10-04-2012] http://www.microsoft.com/cze/education/licence/msdn_academic_alliance/default.mspx

[15] *OCL Grammar.* [01-02-2012] http://www.cs.columbia.edu/~akonstan/ocl/ocl.g

[16] *OCLtoSQL Home Page.* [12-04-2012] http://www.assembla.com/spaces/ocltosql/wiki/

[17] Sam Harwell. *Visual Studio and the ANTLR C# Target.* Version 0.9.

[18] *Tool for PNG to EPS conversion.* [03-01-2012] http://www.socher.org/index.php/Main/PNGToEPSAndJPGToEPSConverterForWindows

[19] Truyen, F. *Mapping MDA Concepts to EA Features*. Sparx Systems, 2005.

[20] *Tutorial: Create your first C# Enterprise Architect addin in 10 minutes*. [11-12-2011] http://geertbellekens.wordpress.com/2011/01/29/tutorial-create-your-first-c-enterprise-architect-addin-in-10-minutes/

[21] *WiX EA Addin deployment*. [10-04-2012] http://geertbellekens.wordpress.com/2011/02/23/tutorial-deploy-your-enterprise-architect-csharp-add-in-with-an-msi-packag

[22] *WiX in VS 2010*. [10-04-2012] http://www.rhyous.com/2010/10/15/wix-creating-an-msi-and-deploying-your-first-file/

[23] *WiX Toolset*. [10-04-2012] http://wix.codeplex.com/

[24] Warmer J., Kleppe A.. *The Object Constraint Language Second Edition - Getting Your Models Ready For MDA*. 2nd edition. Addison-Wesley Professional, 2003. ISBN 0321179366.

List of Abbreviations

DDL = Data Description Language
DLL = Dynamic Link Library
DOT = Dresden OCL Toolkit
DOT2 = Dresden OCL Toolkit 2
EA = Enterprise Architect
EBNF = Extended Backus-Naur Form
GUI = Graphical User Interface
IDE = Integrated Development Environment
MDA = Model Driven Architecture
MS SQL = Microsoft SQL Server
MSDN = Microsoft Developer Network
MSDN AA = MSDN Academic Alliance
OCL = Object Constraint Language
PDF = Portable Document Format
PIM = Platform Independent Model
SQL = Structured Query Language
SVN = Subversion (Apache Subversion)
T-SQL = Transact-SQL
UML = Unified Modeling Language
VS = Visual Studio
WiX = Windows Installer XML
XML = Extensible Markup Language

A. CD Content

- Bin

  OCLtoSQL.msi OCLtoSQL installer.

- Documantation

  thesis.pdf The Master Thesis.

  refman.pdf Generated documentation.

- Examples

  OCLExamples.eap EA project with some examples to run the plugin on.

- Source Code

  Master Thesis Tex source code needed for the thesis compilation.

  OCLtoSQL Source code of the OCLtoSQL plugin.

  Prototypes Source code of the prototypes used for Analysis.

  - GetDDL
  - GetOCL
  - TryToParseOCL

B. OCL Grammar for ANTLR

```
/* Inspired mainly by http://www.cs.columbia.edu/~akonstan/ocl/ocl.g */

grammar OCLConstraint;

options {
    language=CSharp3;
    TokenLabelType=CommonToken;
}

//End after first mistake (no recovery)
@members {
protected override object RecoverFromMismatchedToken(IIntStream input,
int ttype, BitSet follow)
{
        throw new MismatchedTokenException(ttype, input);
}
public override object RecoverFromMismatchedSet(IIntStream input,
RecognitionException e, BitSet follow)
{
        throw e;
}
}

@rulecatch {
catch (RecognitionException e)
{
    throw e;
}
}

@lexer::namespace{TryToParseOCL}
@parser::namespace{TryToParseOCL}

public
constraint
  : 'inv' COLON logicalExpression EOF
  ;

logicalExpression
  : relationalExpression
    ( logicalOperator relationalExpression )*
  ;

relationalExpression
  : arithmeticExpression
```

```
    ( relationalOperator arithmeticExpression )?
  ;

arithmeticExpression
  : unaryExpression
    ( arithmeticOperator unaryExpression )*
  ;

unaryExpression
  : ( unaryOperator postfixExpression )
  | postfixExpression
  ;

postfixExpression
  : (STRING | NUMBER) => literal
  | NAME ( (DOT) => DOT NAME
| (RARROW NAME LPAREN NAME (COMMA NAME)* BAR) =>
  RARROW NAME LPAREN NAME (COMMA NAME)* BAR actualParameterList RPAREN
| RARROW NAME LPAREN actualParameterList RPAREN
)*
  ;

literal
  : STRING
  | NUMBER
  ;

actualParameterList
  : (logicalExpression (COMMA logicalExpression)*)?
  ;

arithmeticOperator
  : addOperator
  | multiplyOperator
  ;

logicalOperator
  : 'and'
  | 'or'
  | 'implies'
  ;

relationalOperator
  : EQUAL
  | GT
  | LT
  | GE
  | LE
```

```
  | NEQUAL
  ;

addOperator
  : PLUS
  | MINUS
  ;

multiplyOperator
  : MULT
  | DIVIDE
  ;

unaryOperator
  : MINUS
  | 'not'
  ;

//////////
//Lexer://
//////////

WS
: (' '
| '\t'
| '\n'
| '\r')
{ $channel=Hidden; }
;

LPAREN : '(';
RPAREN : ')';
LBRACK : '[';
RBRACK : ']';
LCURLY : '{';
RCURLY : '}';
COLON : ':';
DCOLON : '::';
COMMA : ',';
EQUAL : '=';
NEQUAL : '<>';
LT : '<';
GT : '>';
LE : '<=';
GE : '>=';
RARROW : '->';
DOTDOT : '..';
DOT : '.';
```

```
POUND : '#';
SEMICOL : ';';
BAR : '|';
ATSIGN : '@';
PLUS : '+';
MINUS : '-';
MULT : '*';
DIVIDE : '/';

NAME
    : ( ('a'..'z') | ('A'..'Z') | ('_') )
        ( ('a'..'z') | ('0'..'9') | ('A'..'Z') | ('_') )*
  ;

NUMBER
  : ('0'..'9')+
    ( { input.LA(2) != '.' }? '.' ('0'..'9')+ )?
    ( ('e' | 'E') ('+' | '-')? ('0'..'9')+ )?
  ;

STRING : '\''
(
( ~ ('\'' | '\\' | '\n' | '\r') )
| ('\\' ( ( 'n' | 't' | 'b' | 'r' | 'f' | '\\' | '\'' | '\"' )
| ('0'..'3')
  (('0'..'7')
('0'..'7')?
  )?
| ('4'..'7')
  (('0'..'9')
  )?
) ) )*
'\''
    ;

SL_COMMENT: '--' (~'\n')* '\n'
{$channel=Hidden;}
        ;
```