Lund Institute of Technology
Department of Computer Science
Master Thesis Project, 20 points

LUND INSTITUTE
OF TECHNOLOGY
Lund University

Automatic code generation with Roundtrip Engineering

by
Ola Hansson
and
Anders Svensson

2003-09-15

Master thesis project of the requirements for the degree of
Master of Science in Computer Engineering
at Lund Institute of Technology

Supervisors:
Lars Bendix, Department of Computer Science at Lund Institute of Technology
Daniel Eriksson, Software Architect from the assignor

Abstract

During software development of distributed applications, which use the same database, there will be numerous similarities, such as structure and format. If different programs perform the same task, a function in the program, there will also be similarities in the code describing this in the application. If you can identify these functions while designing the applications and reuse the code, you could probably save a lot of time not having to rewrite the code in every new application. An insecure solution is that the developer manually inserts reusable code in the new application. The problems the new developer will face are to find the useful code in a large prewritten application and decide if the code will work in the new application. This will probably take longer than to rewrite the code.

A better way would be if you had a set of functions, and from your Use case description identify these and automatically generate them into your project. If this could also update the project model, you would have a very efficient and manageable tool for software development. The goal of this report is to examine if this is possible using Rational XDE combined with Visual Studio .NET, and how it could increase the efficiency of software development.

Preface

This master thesis was written during the period April – September of 2003 by Ola Hansson and Anders Svensson, Computer Science students at Lunds Institute of Technology. The thesis is intended for students studying Computer Science and other people with skills in object oriented programming, development environments like Visual Studio and model driven software engineering. The user manual, which can be found in *Appendix 2*, is mostly intended for the assignor.

We would like to thank our supervisors Daniel Eriksson and Lars Bendix, and all other people who helped us creating this thesis.

Content

1 Introduction

The task is to investigate if it is possible to develop a way to automatically generate code into a Visual Basic .NET project that is constructed according to a certain basic structure. To make this a natural part of the development process, which is done using Unified Modeling Language (UML) models, the code generation should also be model driven and not add unnecessary complexity to the development process. The code generation should be started by editing the model and when code has been generated into the project, the model should be automatically updated. The user should be able to create a base for the application by examining the Use case descriptions and identifying the functions that can be generated.

For modelling, Rational XDE will be used and we were to investigate if it is possible to create a system for generating code with this modelling environment and if the tools included in Rational XDE can be of any help to us doing this.

1.1 The assignor's idea

The assignor is an international company in the retail business with dealers, suppliers and storage units all over the world. The company wants to make their information exchange more efficient. In the current system every division of the company has its own database with information concerning that division. If they need to exchange information with another division of the company a connection to this division has to be established. This is not a good system when it comes to efficiency and reliability. Because of this the company is developing a new system for handling the information. This is a client-server system with centralised servers that store the information and clients which the divisions use to access the information.

The server applications are already implemented and now they are about to start developing the client applications. Different client applications will be used to access the same data on the server as described in *figure 1.1* below.

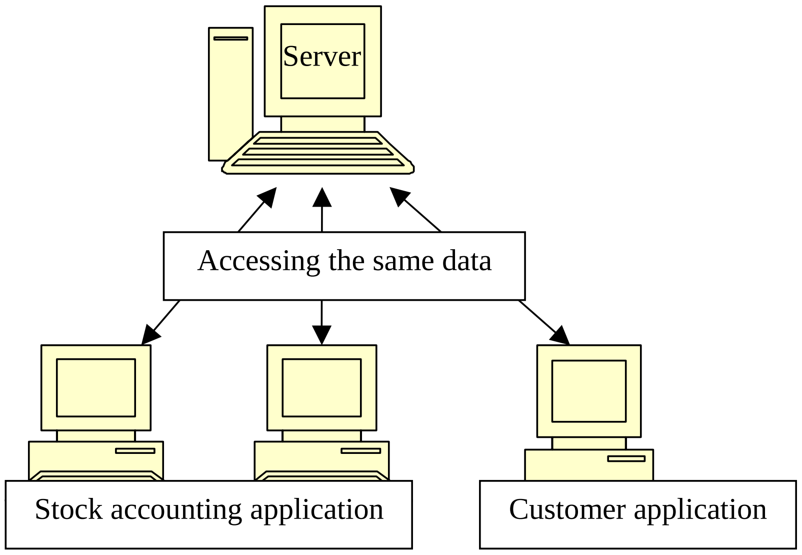


Figure 1.1 A client-server system with different client applications accessing the same data on the server.

The stock accounting application may need to check how many items of a certain kind a store has, to control when to order more of the item. A customer application may need the same data to give the customer information about where the customer can find the item.

Other similarities in the applications will be standard functions such as control devices for user data, error handling and so on. When developing the applications, these similarities will result in similar code in the different client applications. This gave the assignor the idea that maybe these parts of the code could be automatically generated and if this could be done in a smart and efficient way, maybe development time could be reduced. This approach would also give a more uniform code, which could increase the correctness and reduce the time used for maintenance work.

The client applications are to be written in Visual Basic .NET, and the tools the assignor wish to use are Rational XDE Developer, which is a modeling environment for creating UML models, combined with the development environment Visual Studio .NET. While developing the server applications the predecessor to XDE, Rational Rose was used to automatically generate code, and now the assignor wants to know if XDE could be used for generating the Visual Basic code in the client applications.

1.2 Goals

The main goal of the assignment is to investigate the possibility to reduce development time by automatically generating code into a Visual Basic .NET project and if possible, develop a system for doing this.

By investigating this, we would like to find answers to a couple of questions:

- Is it possible to develop a model driven code generator with Rational XDE? Does XDE contain useful tools for this matter? Is the pattern tool within XDE a good way of generating code?

- Should the code generator be integrated in Visual Studio .NET as an add-in or should it be a separate program that runs independently? What benefits can be gained in the different solutions?

- If it is possible to develop a functional code generator, is there anything to gain by using it or does it introduce other problems for the developer, like creating the code that should be generated and rules on how to write source code?

- What kind of problems can arise when using a code generator and how can these be avoided? If a problem appears during code generation, how should the application handle this?

- How much extra work must be done after adding code to the project?

1.3 Outlines of the thesis

Chapter 1 - Introduction
A description of the goals and a description of the assignment.

Chapter 2 – Background
Theories and tools used to investigate the assignment.

Chapter 3 – Solution
Description of how the assignment has been solved and which complications have arisen.

Chapter 4 – Evaluation and discussion
Feedback and discussions about the assignment.

Chapter 5 – Conclusions
The conclusions we made of our development work.

Chapter 6 – References
Information references that we have used in this report.

Chapter 7 – Appendix
This section contains the appendices, which are a user manual for our system and an example of an XML document.

2 Background

This chapter considers the tools and theories used in solving the assignment. We also describe our first idea on how to solve the assignment and use this to motivate the other sections in this chapter. The description of the idea gives an introduction to the content and terms used in the report. How this idea later evolved into the solution of today is described in the chapters to come.

For further information about the tools used, web references are given in chapter 6.

2.1 The first draft

When we approached this problem we thought of an idea where all software development is done from one base system. By base system we mean an implemented design solution common for all applications that will be developed. When studying the planned applications, the developer should identify the equal functions in the different applications, for example retrieving some specific information from a server. If there is a function that exists in several of the applications, there may be some profit in automatically generating this function into the application when creating it.

Because of this our idea, which is shown in figure 2.1, was that the developer by editing the model should be able to add these functions to the application being developed and receive executable Visual Basic .NET source code. To make this solution possible we need a way to add the functions to the existing base system for all the applications. The result of this should then update the model to include the code for the function.

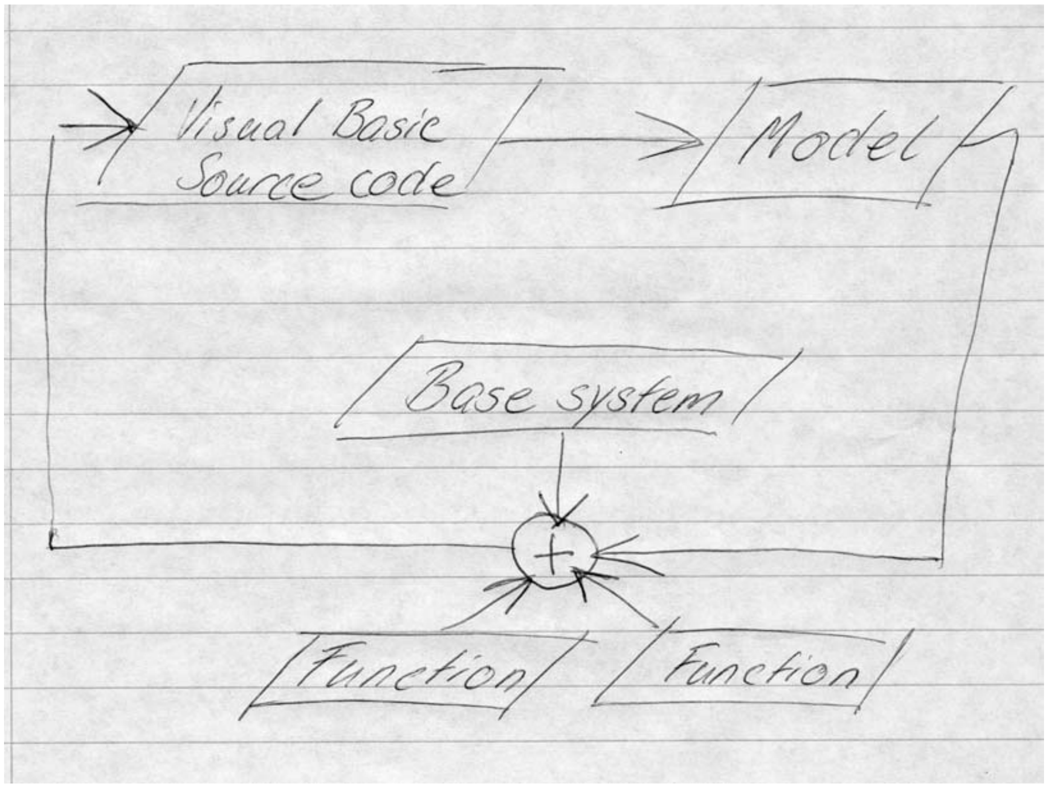


Figure 2.1 The first draft of the code generating model

This solution leads to that the developer has to create a set of functions that can be generated for the base system. Because of this we need to find a way to represent the functions in what we call function files. These files must contain the source code needed to describe the function, and also a way of describing where the code should be placed. To obtain this we used the text format XML, which is described briefly in section 2.5. For editing the source code we used a namespace in the .NET Framework called CodeDOM which is described in section 2.6. More details on how these tools were used in our solution can be found in the next chapter.

2.2 Microsoft Visual Studio .NET

Visual Studio .NET is Microsoft's latest release of their development environment and is the sequel to Visual Studio 6. This program includes the .NET framework, which is described in section 2.3. Visual Studio .NET is a very useful development environment with tools for simplifying development of Windows Forms, Web Forms and XML based data. Visual Studio.NET also converts some writing errors (for example upper/lower case errors) and uses IntelliSense to give the user a better general view of the namespaces, functions and so on, that can be used while developing code for an application. Another function that this development environment has got is the usage of break points, used for debugging. This together with the possibility to actually see what kind of values the variables in use of the application have, made the debugging fairly easy.

2.3 The .NET framework

Microsoft developed the .NET framework in the year 2002 (current version 1.1 was released April 2003). The main purpose of the .NET framework is to simplify communication between coded applications. The .NET framework is based on two main parts, the Common Language Runtime (also known as CLR) and the .NET framework Class Library. The .NET framework is distributed with the Windows Server 2003. It can however be included in the applications, if the developer has used the redistributed version of the .NET framework Software Development Kit, to allow all platforms to execute the applications. Another way to make the framework able to execute .NET applications is to download the framework by itself.

2.3.1 Common Language Runtime

The CLR is a runtime agent which controls the execution of a .NET application. This agent provides the following services:

- Memory management – a garbage collector is introduced to avoid memory leaks and invalid memory references.
- Thread management – all threads must be isolated from each other so that one thread does not crash the whole system.
- Code management during runtime – through the Just-in-time compilation between Microsoft Intermediate Language and native machine code the CLR has enhanced the performance of executing code. This resembles the way Sun uses byte code in their programming language Java.
- Security – file access operations, registry access operations and other sensitive functions are being supervised.
- Exception handling – all exceptions, for example if an application tries to open a file which does not exist, are correctly managed to prevent the system from crashing.
- Interoperation – gives the developer the possibility to use COM components and DLLs.

2.3.2 .NET framework Class Library

The .NET framework Class Library is a large object oriented library with all the conceivable classes, interfaces and value types a developer can use to gain access to system functionality. This library has a hierarchic structure with naming syntax using the dot notation to separate namespaces, just like Java’s packages. All functions, enumerations and so on have detailed information on how the functions work and what type of arguments the function needs to be executed.

2.4 Rational XDE

Rational XDE is a modeling environment for creating UML models. XDE uses the UML 1.4 standard to describe the models, and is the sequel to Rational Rose. The Rational XDE family contains three different versions; XDE Modeler, XDE Developer and XDE Tester. All the programs in the XDE family can be integrated in Visual Studio .NET and IBM WebSphere Studio Application Developer or run as stand alone applications.

Rational XDE Modeler is used to produce platform independent UML models of software architecture.

Rational XDE Developer is a complete development environment for creating applications from UML models. The code generation in Developer is performed by a system which Rational calls Roundtrip Engineering. Roundtrip Engineering works in both directions, which means that you can both generate code from your model and generate models from your code.

Rational XDE Tester is an application used for testing and debugging. This is also included in Developer Pro version.

As mentioned earlier we were using the Developer version as an integrated part of Visual Studio .NET. This adds a Model Explorer and a Model View to Visual Studio .NET. The Model Explorer displays all the elements in the model of the project as a tree structure. In the Model View a more classical UML class diagram is displayed, showing dependencies between elements and so on.

Rational XDE includes a tool for applying design patterns and code templates to automate repetitive coding tasks. XDE comes with all Gang of Four patterns and there is also a possibility to define your own patterns.

Rational XDE was not yet officially released when we started our project, but thanks to our assignor’s good relationship to IBM, we had an evaluation version of the program available to us. During our work a new version was released, Rational XDE v.2003.06.00 which included a documented application programming interface (API) called RXE (Rational XDE Extensibility). This would according to Rational extend the power of patterns and code templates. We changed to this version during our work on the project, but if it had been available from the beginning of our project we may have chosen a different approach on how to solve the task. This will be discussed later.

2.5 XML

XML is short for Extensible Markup Language and is a simple and flexible text format using tags and is often used to describe self defined data structures. The basic idea in XML is to define your own tag language that corresponds to your needs.

XML consists of a set of elements which holds the data. An element contains other elements and attributes. The elements are called nodes and their attributes can only be a set of user defined values. All elements placed under a node are called children. The XML namespace in the .NET framework contains classes to handle XML documents with functions such as parsing, editing, validating and generating.

To validate a XML document, it has to follow the user defined rules of an XML schema. The XML schema defines the structure of an XML document and declares all possible values and types of the attributes in each XML node. The XML schema has, apart from the XML document, a clear defined set of rules describing how it should be designed.

2.6 CodeDOM

CodeDOM stands for Code Document Model and is a namespace in the .NET framework. This namespace is used for building up source code elements, called CodeDomElements in the CodeDOM, as a type of graph that can be compiled or used to generate source code from. CodeDOM is .NET language independent, so generating source code for Visual Basic .NET is as easy as generating source code in any other .NET language. All .NET languages build the CodeDOM structure in the same way, the only difference is which type of code generator, called `CodeProvider`, is used for generating the source code. In this thesis CodeDOM is used to generate Visual Basic .NET source code, i.e. the `VBCodeProvider` is used. When you are generating source code you can choose which type of programming style you want to generate to get it just the way you prefer.

2.6.1 CodeDOM structure

On top of the CodeDOM hierarchy is the `CodeCompileUnit`. This `CodeCompileUnit` contain one or more `CodeNamespace` (all .NET languages can not handle multiple CodeNamespaces or CodeTypeDeclarations in a CodeCompileUnit). Classes are represented by `CodeTypeDeclaration`, which is also used for structures, interfaces or enumerations, and contain one or more CodeTypeMembers. `CodeTypeMember` is a base class for methods and fields. `CodeTypeDeclaration` can also contain `CodeMemberField` which defines an attribute to the class. In this way the structure of the source code is built and how this is used in our system is described in section 3.4.

3 Solution

This chapter is a description of our solution to the problem and how we came up with this solution. We start by describing the main idea in section 3.1 and 3.2, and then we go into details in section 3.3 to 3.6 where the solutions to specific problems are analysed.

3.1 Developing the idea

As we were learning about the tools, we started evolve our idea on how to solve the assignment. The importance of adding functions to a created base system was understood. To do this without causing damage to the system, the base system files as well as the function files would need some form of representation, which was easy to work with. The first idea we had was to place tags with information about where to put namespaces, classes, methods, attributes and so on in the source code files and then insert the code in the right place according to the tag information. This method was soon abandoned because it would be both complex and complicated to keep this file uniform, considering searching and changing the development code. At that time we got to know about CodeDOM. However we still had the same problem if the code of the base system would be changed by adding a function. CodeDOM is not a good way of representing data; it is just a tool for generating source code files or to execute files while already running an application. To find a good way to represent and store the data in some form of code, XML caught our interest. After investigating how XML works, this way was chosen to represent the code because this matched our purpose well. How source code is described in a XML document is found in section 3.3. When the implementation of the classes using CodeDOM and XML was done, we focused on solving the problem with updating the model of the generated system automatically and how to display this in the model view of Visual Studio .NET.

3.2 The Roundtrip model

After coming to the conclusion that XML could be an appropriate way of describing source code, we started to develop our first idea from chapter 2. We wanted the developer to be able to add functions both before and after that he has edited the source code manually. To make this possible we needed a system where you could always reach a state where it is possible to add functions. This made us start developing a model where we make sure that the application always can be described as XML, which we call the Roundtrip model. The figure below describes the first version of the Roundtrip model

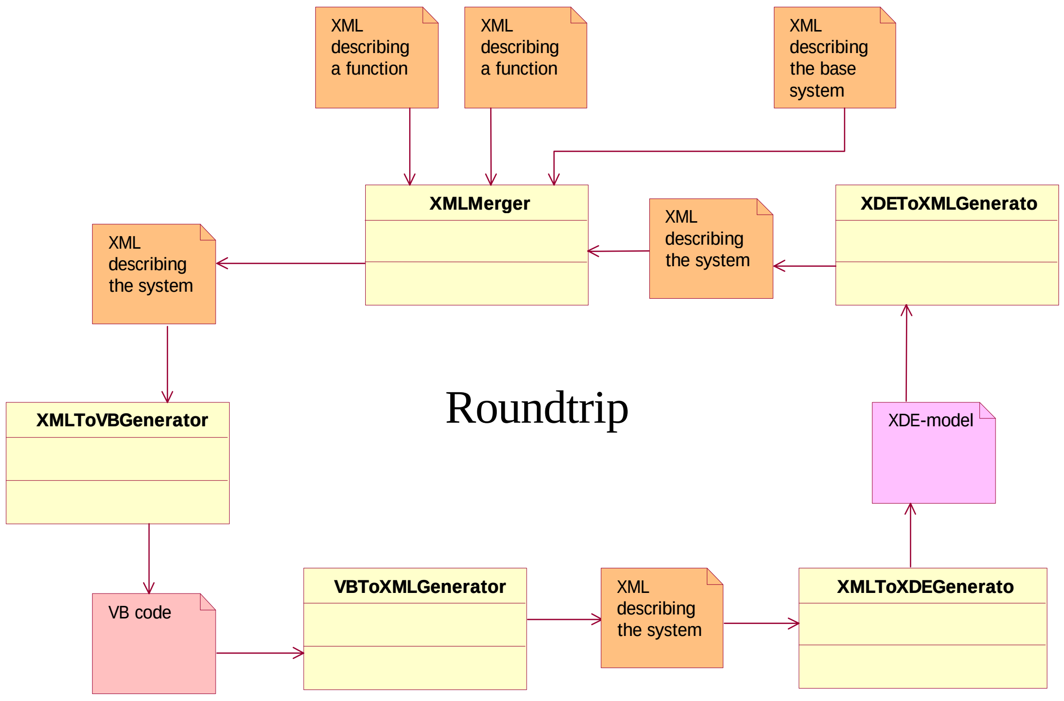

XML describing a function
XML describing a function
XML describing the base system
XMLMerger
XML describing the system
XDEToXMLGenerato
XML describing the system
Roundtrip
XMLToVBGenerator
XDE-model
VB code
VBToXMLGenerator
XML describing the system
XMLToXDEGenerato

Figure 3.1 The original Roundtrip model

The generation of the functions is performed by the XMLMerger and is done by merging XML documents of the base system or the current system, and a function file. As output from the merge operation, an XML document describing the system with the function added is produced. How this merge operation is performed is described in section 3.6.1.

The next step is to transform this XML document to Visual Basic .NET source code that can be further developed. This is done in the XMLToVBGenerator as described in section 3.4. The idea was then that the source code, after editing, should be transformed back to XML to be able to create the model of the system through the XMLToXDEGenerator. This would give the user the possibility to further develop the application through the Model View. Finally the XML document of the system would be recreated from the model and if any function had been added in the model, this would be merged in and the source code would be updated.

The idea we had about controlling the function generation by letting it be done by the tools included in the Rational XDE application programming interface (API) called RXE, did not appear to be possible and instead we created a graphical user interface (GUI) to guide the user. From the GUI the user can add existing functions, create new functions and create a new base system.

To create the model from XML descriptions seems to be an unnecessary procedure, since there is a functional way to go from Visual Basic source code to the model by assistance from Reverse Engineering. This function parses the source code and builds a complete model from that. Besides, it is really unnecessary to go from source code to XML representation just to update the model. Unfortunately it also turned out to be impossible to automatically update the Model View from the source code, since the possibility to change the model was left out in the API of Rational XDE. This made us change the roundtrip model and let the connection between source code and model be handled by the tool in Rational XDE. Nevertheless, there may be a possibility to control the generation of functions by assistance of RXE, but we did not have the time to investigate if this is possible. The revised model can be seen below in figure 3.2

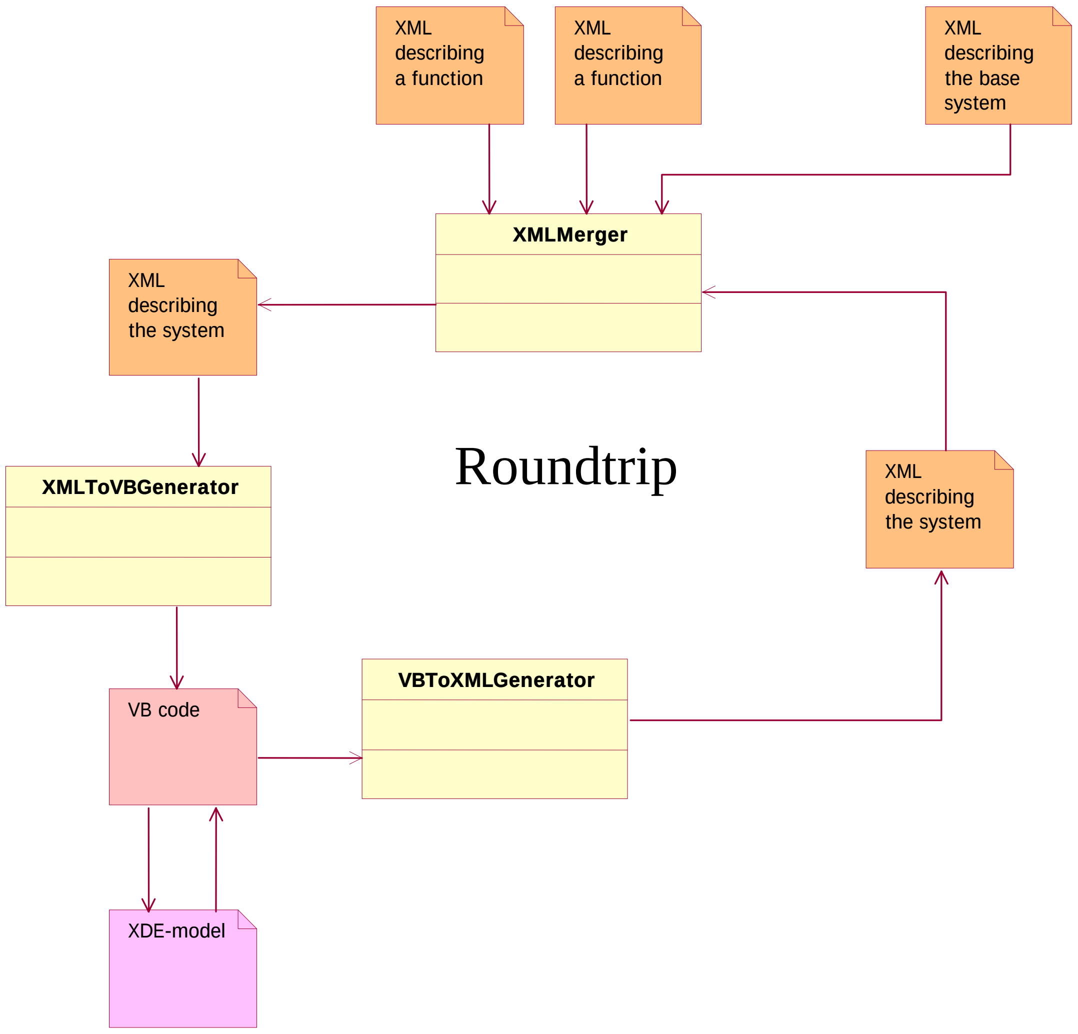

XML describing a function
XML describing a function
XML describing the base system
XMLMerger
XML describing the system
Roundtrip
XMLToVBGenerator
XML describing the system
VBToXMLGenerator
VB code
XDE-model

Figure 3.2 describing the current roundtrip model

3.3 Source code as XML

To be able to create the Roundtrip model, we needed a way to compare and merge source code without having to parse it each time. We also needed a way to store the code for the functions in a clever way. The solution to this was to describe the source code as an XML document. We created a tree structure, similar to the one in CodeDOM, describing the source code. XML also gives the possibility to have more information about the functions in the XML documents. The reason we chose XML is because of its flexibility and because good tools for parsing XML documents exist. An example of a small XML document and the corresponding Visual Basic .NET source code is found in *Appendix 1*.

3.3.1 The Code node

The `Code` node is the base node for the XML document representation of the source code. It is from this node the tree structure describing the source code is built.

```
<Code>
  <Namespace Name="ClientApplication">
    <References>
      <Reference Name="System" />
    </References>
    <Classes>
      <Class Name="Application"> ...
```

3.3.2 Namespace

The next node in the hierarchy is the `Namespace` node. This node represents the namespace in Visual Basic .NET and has the name of the namespace as node attribute. There can be several `Namespace` nodes beneath the `Code` node but the most common is just one namespace.

3.3.3 References

The `Reference` node is placed under the namespace node and contains the references needed in Visual Studio .NET to make the code work. A reference can be made to the following types of components:

- .NET class libraries or assemblies
- COM components
- Other class libraries of projects in the same solution
- XML Web services

3.3.4 Classes

The `Classes` node is also placed under the `Namespace` node and contains nodes describing the different classes in the namespace. Every `Class` node has the class name as an attribute and if the class is a subclass, the name of the class which it inherits is included as an attribute. If the class has a description text this will also be included as an attribute. The structure of the `Class` description can be seen below

```
...
  <Classes>
    <Class Name="Application">
      <Imports>
        <Import Name="System.Error" />
      </Imports>
      <Attributes />
        <Attribute Name="TheGUI" Type="GUI" Access="Private" />
      </Attributes>
      <Properties />
      <Methods>
        <Method Name="Main" Access="Public Shared">
          <Parameters />
          <MethodCode>
            <Line Code="Dim theGUIControl As New GUIControl()" />
          </MethodCode>
        </Method>
      </Methods>
    </Class>
  ...
```

3.3.5 Imports

The first node below the class node in the XML document is the `Imports` node which contains nodes describing the imports statements in the code.

3.3.6 Attributes

The next node in every class node is the node containing the attributes of the class. Every `Attribute` node is described by three attributes; its name, type and access.

3.3.7 Properties

After that the description of the properties follows. A property is a method for handling an attribute in class. The `Property` node contains a `Get` and/or a `Set` method. These are described as nodes under the `Property` node. The code in the `Get` and `Set` methods is put as attribute in one node for every line of code as shown below.

```
...
  <Line Code="MainWindow = New MainForm()" />
  <Line Code="MainWindow.Activate()" />
...
```

3.3.8 Methods

The methods are described in `Method` nodes placed under the `Methods` node in the class. A `Method` node contains the name, access, type, description and event handling as node attributes. The parameters for the method is described by nodes containing the names and types of the inparameters under the `Method` node The code for a method is described in the same way as in the properties and is placed under a `Methodcode` node.

3.4 Code generation from XML using CodeDOM

When you are about to generate Visual Basic source code from the XML document you try to find the code node in the document. From this node the XML document is parsed node for node, first finding the namespaces and then moving further into the tree structure to build the classes.

To create a class a wrapper to the CodeDOM is used, which also builds a structured namespace containing the current class by assistance of CodeDOM. CodeDOM completes the code generation by generating the Visual Basic source file.

To build a CodeDOM structure, the first thing you have to do is to create a `CodeNamespace`. To this you add classes and `Imports` statements, which will apply to the whole file. Then you add attributes, methods and all other information the file has to contain. After that you add comments, inparameters, return types, method code and etc to the methods. This makes CodeDOM simple and flexible to use.

3.5 Creating a XML document from Visual Basic source code

To create an XML document the Visual Basic source code has to be parsed and interpreted. Our first approach was to try to find a prewritten parser, which we also did in the CodeDOM namespace in the .NET framework class library. Unfortunately this parser was not yet implemented to handle Visual Basic .NET source code; it was only an interface that could be implemented. At that time this was not of current interest and we decided to write our own, less advanced parser which would fulfil our needs.

3.5.1 The Parser

The parser works as described by the flowchart in *figure 3.3*.

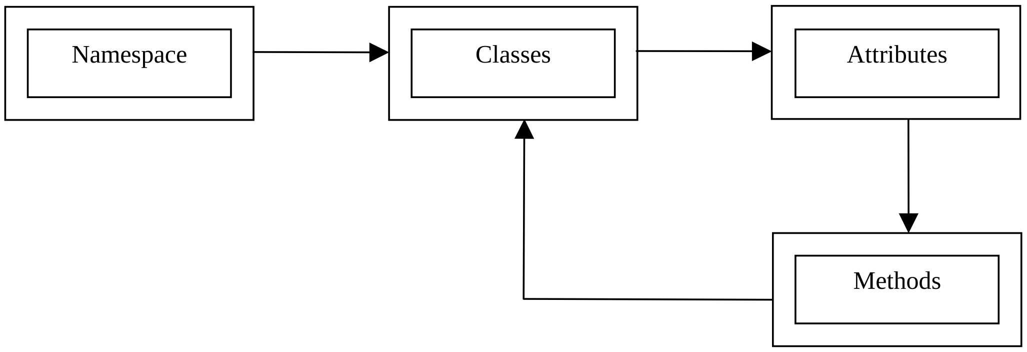


Figure 3.3 The flowchart of the parser

1. Namespace – this is the first thing in the file that the parser searches for. If no namespace is found, a `Messagebox` is generated asking the user to prompt the wanted name of the namespace. All files must contain a namespace because CodeDOM generates code on the basis of the namespace. Note that there can be only one namespace in a file.

2. Classes – after the position of the file has been reset (i.e. so the search will start from the beginning of the file. This is not actually necessary if the parser finds a namespace in the file but is a safety measure) the first class declaration is found. Only `Public Class` statements are considered to be a correct class declaration in this version of the project, more about this in the section 4.2.1. In this search it is also taken into consideration the possibility that a class can have attributes connected to them. If this is the case, the class attribute has to be placed on the line above the class declaration ending with an underscore or on the same line just in front of it. The parser also makes a control to check if the class inherits another class, which will be stated on the line below the class declaration. Then the parser checks if a comment belonging to this class is stated on the line above the class declaration or in case of class attributes, above this. All comments in the project can consist of one or more lines. Before the parser starts to search for members (that is, attributes, methods and so on) of the class, it searches for the `Imports` statements. This should probably have been done during the search from the beginning of the file to find the class declaration, because the `Imports` statements apply to the whole file (multiple classes are allowed in the same file and namespace) and not to the classes actually. Unfortunately this was noted too late to change the code and since it works both ways and the parsing time is so short, we did not change this.

3. Attributes – the way to find the attributes is to start searching from the position where the class was declared. First all the private, then the public and finally all the friends attribute are searched for. Of course the developer has the possibility to increase the functionality of the application, meaning that more attribute types would be recognized as attributes by implementing this. The actual search is performed by searching for code lines that consist of the words that the developer defines as possible attribute words, for example `Public` and `As,` if a public attribute is desired. This search starts from the current position in the file and continues to the line which ends a class declaration, i.e. `End Class`. When a code line is found with these words included, a control is made to check if the code line really is an attribute declaration and not just a code line with these words in it. The accessibility is stored and the type of the attribute is investigated and possibly converted to a special system type (i.e. the system type of Integer is Int32) and stored for sending further on. Attributes can not have comments though.

4. Methods – the methods can have attributes belonging to them, just like class declarations. The attribute has to be placed on the line above with an underscore at the end of the line or in front of the method declaration, the same way as class declarations. Then the complicated work follows to divide the method declaration in case of inparameters and their types and names, return statements and so on. Before sending the information on, the parser also saves the comment belonging to the method. The methods the parser can handle are:

   Dispose methods are always auto generated when developing Forms, and look like below,

```
Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
```

   The string the parser is using to define that a dispose method is being used is

```
Protected Overloads Overrides Sub Dispose
```

   on the current code line.

   `Main` methods are recognized by the words `Public`, `Shared`, `Sub` and `Main` on the current line to investigate.

   Constructors have to consist of the words `Public`, `Sub` and `New`. This means that no constructors are able to return anything, but this does not seem possible to do in Visual Basic .NET.

`Handles` methods are methods which handle a raised event, for example,

```
Public Sub ButtonClick() Handles ButtonClick
```

handles the event of a button click when this is raised in the application. The method is recognized as an ordinary method, with the extension of the string `Handles.`

Ordinary methods consist of the following type of methods, `Public Sub`, `Private Sub`, `Public Function` and `Private Function`. Naturally the application can be developed to handle several special cases of method types and the amount of changes in the code is not too excessive, but the way the parser splits up the methods has to be changed.

Property methods are handled in a different way from than the methods above. A property is a method that the user can use to read and write values of private declared attributes from other classes. A property looks as follows:

```
Public Property Line() As String
      Get
            Return mstrLine
      End Get
      Set(ByVal Value As String)
            mstrLine = Value
      End Set
End Property
```

These properties can be read only or write only and then the `Set` respectively the `Get` lines are excluded. When CodeDOM generates properties the parameter declaration after the `Set` is not included for some reason. The `Get` and `Set` code (between the `Get`/`Set` and `End Get`/`Set`) are parsed straight off, but it is assumed to be correct and the only thing the parser does is to fit the indentation of the code in these lines. So if comments are made in this section the parser does not care and just adds the code line as it is.

5. The parser is now finished with parsing one class and the work goes back to step 2 and starts again looking for more classes in the file from the position where the previous class ended. When the parser has reached the end of the file the parsing is finished.

It is very important to tell apart the main methods, ordinary methods and constructors because CodeDOM uses different methods to make the source code generation complete.

3.6 Functions described in XML

The functions are stored as XML documents. Every function file is connected to a certain base system. Because of this, the function files are placed in the same folder as the XML document describing the base system corresponding to these functions. The program then searches this folder to find out which function files are available to the selected base system.

The difference between the function files and the XML documents describing the source code for the system is that the function files have an extra node describing the function. The node contains the name of the function and a description of it. It is important that the description of the function contains all the information needed by the user, such as; how the function works, how to call the function and where the code will be placed.

3.6.1 Merging function files and system files

To add a function to the application, which is being developed, the code for the function has to be merged with the code for the application. This is done as follows:

- The Visual Basic .NET files containing the source code for the application are removed from the project. The reason for this is that if the files still are a part of the project, they can not be edited by any other program.

- The next step is to parse through the files and create an XML representation of the code. This is done by the code parser described in section 3.5

- After that, the XML document describing the function is merged with the newly created XML document describing the current system. The result of the merge is the system with the new function added and is saved under a new name. The old file is later used when validating method code that has been changed during the merge.

- The merge is done recursively, depth first, by comparing the function file with the system file. Every node in the function file is compared with the corresponding node in the system. If the node can not be found in the system, the node and everything beneath it are added to the system. If the node is found but there are differences in the underlying nodes, the nodes under it are examined until the differences have been found. Otherwise the program moves on to the next node on the same level in the function file. The nodes containing source code are handled a little bit differently. If a method is found where the only difference between the system and the function is the code inside the method, the methods are merged by adding the code from the function after the code from the system. Then the user gets an opportunity to validate and edit the code if needed.

- Finally the new system file is used to create new Visual Basic source code files the way described in section 3.4. The files may look a little bit different than they did before the function was added even if the code in the file has not been changed. This is caused by the fact that all the code is transferred to XML format and when regenerating the files, CodeDOM decides the structure of the files. These files are then inserted into the project and the function has been added to the application.

3.6.2 Creating function files

There are no rules for what kind of functions can be generated. A function can be everything from a line of code to a practically complete program, as long as they are not dependent on anything but the base system they were created from. Neither of these is a good way of using the program since they will not reduce development time nor make the development process easier.

The function files are produced by implementing the code for the function, starting from the code for base system. After that the changed system is compared to the original base system, and the differences between the two create the function file. The comparison is made in the same way as the merging of a function and a system file when adding a function to an application. Nodes are added to the function file when the node or any of its children differs from the base system. If the node does not exist in the system file, the node and the tree structure under it are added. If the node exists and there are differences beneath it, the node is added and the structure under it examined. Otherwise if a node exists, the node is ignored and the program moves on to the next node.

A node containing the name of the function and a description text is added to the XML document and is used when presenting the function to the user.

The benefits of developing the function files from the base system are that it is easy and does not require any extra skills for the developer. Since the function file is developed from base system, the function will work correctly with that base system. The disadvantages are that the function can not be tested with any other code while developing them without removing all the code that is not a part of the function code before generating the function file.

4 Evaluation and discussion

This chapter is an evaluation of our solution. In the first section we try to answer and discuss the questions asked in section 1.2. The following section contains problems we encountered during our work and limitations in our solution. In the final section we discuss the benefits and disadvantages of using our solution, enhancements that could be added to it and alternative solutions to problems.

4.1 Results

The assignment had two main goals, investigating if it was possible to develop a code generator with the selected tools and if this was possible, examine if it could be done in a cost efficient way. These goals led to a couple of questions that had to be answered.

4.1.1 Tools

A main objective of our work was to evaluate if the tools the assignor wanted to use for the assignment were appropriate for the matter.

We wanted to know if it was possible to create a model driven code generator using Rational XDE. Using the pattern tool included in XDE is a possible way to obtain this. The problem though is that the pattern tool is not created to generate code into an existing project. It is designed to apply design patterns on a project and would be more suitable to give projects the same overall structure. In the pattern tool there is a possibility to connect code templates to a pattern. Using these would be a possible solution, but writing the code templates is complicated if the function you want to implement needs more code than just one method, which is likely.

The graphical user interface in our implementation is partly a replacement to the model driven approach. We were hoping that the programming interface RXE would give us the possibility to affect the model and how it is presented to user before and after a function has been added, but RXE did not allow the user to change the model. This made the model driven approach impossible.

We chose to design the program as an add-in to Visual Studio .NET, to avoid adding unnecessary complexity to the development process. This solution also meets our demands on usability and enables a future extension of the integration. The program should be an easily accessed tool to use when setting up and configuring a project. An external program would have forced the user to import and export the source code files manually. This choice also made it possible to use internal calls in Visual Studio .NET and Rational XDE without having to worry about calling the program by operating system calls.

4.1.2 The code generator

The main reason for developing a code generator is to make development faster and easier. If our solution fulfils this depends on a couple of factors:

How many different applications are being developed from the base system? There is no point in using the system if the number of applications is low.

Do the applications have clearly defined functions that are used in several of them? Finding the similarities in the applications is the main idea behind this program, so if there is no or only a few similarities there is no point in using the program and the development should be done using a more classical approach.

Can the functions be implemented so that they will not be dependent of any other code than the one included in the base system? This is in a way handled by the system we use for creating a function file, where you start from the base system and develop from that.

Since the function files are created in the same way as creating them as a part of the application, the development time for a function should not have to increase. However, the developer has to set time aside for evaluating which functions in the application should be created. Functions should be created if the time it takes to create them is less than the time gained from generating them into the applications instead of rewriting them.

4.2 Problems and limitations

Due to the limited amount of time there was no possibility to build a system that could handle all possible Visual Basic .NET source code. Both the parser and the XML description of the Visual Basic .NET code have limitations regarding which code it can handle. We chose to follow the programming guidelines for the application the assignor wants to develop using the system

4.2.1 Parsing Visual Basic files

The parser has in our release some limitations of how code has to be written and what kind of source code the parser can handle when it comes to different types of methods and attributes. The parser is well adapted to our purpose and specified according to the code examples the assignor has supplied us with.

An example of this is the recognition of different types of classes by the parser. Our system only recognizes public classes. Private classes can only be used within the same definition area as the declaration and is not used too often. Other types of class declarations, which can be used when a class must inherit or must override another class, are not implemented either. The effort to develop the parser to recognize such declarations is minimal, but to make the whole system handle this is more demanding.

The parser also has some limitations when it comes to attribute declarations, the parser can not handle attributes that are initialized with a value at the same line as declaring the variable.

The code is not checked for syntax errors, but is assumed to be correct according to the source code control of Visual Studio .NET. For example if an attribute declaration is incorrect according to our definition of attributes, the parser will not notice the attribute. On the other hand, code within a method is not checked at all, which means that if an attribute is declared incorrectly and later used within a method, an error will occur. Relying on the source code control in Visual Studio .NET may not be a good solution, but sufficient enough for our purpose.

4.2.2 Generating source code

The development possibilities with assistance of CodeDOM are many, but there are some limitations. Everything can not be created the way you want it.

As an example, the following line of code,

```
Public Sub MethodName() Handles ButtonClick
```

can not be directly generated because CodeDOM does not use `Handles` methods the same way as the assignor are used to. The assignor’s construction derives from earlier Visual Basic versions. The problem was solved by means of generating the method as function with the return statement in this example `Handles ButtonClick`, and then changing the generated source code from

```
Public Function MethodName() As Handles ButtonClick
```

to

```
Public Sub MethodName() Handles ButtonClick
```

This could probably be done in a way by using the `CodeSnippetTypeMember`, a representation of a member of a type using a literal code fragment that is included directly in the source without modification. But the problem which occurred was that the `CodeSnippetTypeMember` could not be used as a regular `CodeTypeDeclaration`, which makes adding code lines, method parameters and so on impossible. Unfortunately all efforts to solve this resulted in other problems with the expression, which is why this is not done in an optimal way from a development point of view.

The `Get` statement in the `Property` methods is not generated as the assignor has generated their source code files. The difference is that CodeDOM does not generate an argument for the `Set` statement. This is no problem, since the `Property` works either way. The parameter is declared in the `Property` declaration.

The constructors that are generated are always generated with the code line `MyBase.New` without parentheses. This means that the parser always have to check if that code line already exists in the code of the constructor, and possibly add that code line with parentheses to the constructor to get a consistent appearance of the generated code. Writing a method call in Visual Basic .NET without the parenthesis does not produce any errors but should be avoided for reasons of readability and consistency.

To be able to generate the source code files, CodeDOM needs a namespace to generate the classes from, and results in every Visual Basic .NET file containing a namespace statement. This causes a small problem when the CodeDOM generated files are to be used in Visual Studio .NET. When creating a new project in Visual Studio .NET, a namespace with the same name as the project will automatically be created and all classes will be placed under this namespace. The problem is however, that this namespace will not create any namespace statements in the Visual Basic .NET code. So when the generated files are imported into the project, the namespace described in the source code files will be placed under the one described by Visual Studio .NET, which will result in two nestled namespaces in the project, that may look a bit complicated. We have not found any way around this problem and since it does not affect the functionality of the project, we do not think it needs to be solved.

4.3 Discussion

As mentioned earlier in this chapter, the gain of using our solution is very much dependent on the analysis of the applications the developer wishes to create. Which criteria this analysis should follow is hard to define without testing the system on a large scale and could be an objective of further research.

A major problem while developing functions to a base system is name conflicts. For example `Imports` statements should be used restrictively since `Imports` statements are used to avoid having to write the full path. Because of this, an `Imports` statement which works with the base system can easily cause a conflict when combined with code from another function. Writing functions where methods have to be merged with existing methods should be avoided if possible. To avoid it the methods used in the function should if possible get unique names. An idea could be to use the name of the function as a prefix to the methods.

The developer has to keep in mind that only the name of the reference and not the path is given in the Reference node in the XML document. References to classes should be avoided if it is not sure that user has access to these classes. If this is the case, it is better to include the code in the base system or function file.

The model driven approach could not be achieved the way we tried. A possible solution to this could be to try and generate commands on operating system level, such as mouse events, in the model explorer in Visual Studio .NET. This would give us access to commands in Rational XDE that could not be accessed by the API. This is probably not an easy task for any programmer who is not an operating system expert. It could also be interesting to further investigate the possibilities of the programming interface RXE combined with the code templates tool. We did not have time to do this since this would change our approach to the problem completely. Maybe this would be a way of achieving model driven code generation.

Maintenance work on the base system and function files can not be done without reimplementing the applications created from these files. This is due to the fact that once a function file has been added to the current system it is part of the system and can not be separated. Keeping them separated would solve this problem and give the developer a possibility to update the implementation of a function without having to manually edit all the applications using this function. This would require a more sophisticated solution on how to merge the XML files.

To increase the usability of the system the user interface needs to be improved. In this version this was not our top priority and to make the system useful on a larger scale, enhancements have to be made concerning this.

5 Conclusions

While investigating the possibilities of constructing a system for code generation, we came to the conclusion that making a model driven system, with assistance from Rational XDE, proved not possible at the moment because there is no way to access the model.

If the model driven approach is set aside, there is a possibility to create a functional system for generating source code automatically. If the system is cost efficient is foremost dependent on what kinds of application the user wants to develop. If several of the applications have similarities which can be identified as functions of the applications and these work independent of each other, a profit could be made from using the system. When constructing a set of functions to use for generating into applications, functions should be included in the set if the time it takes to create the function is less than the time gained when generating it.

The problems that arise when developing a system for generating code mainly depends on handling source code. Source code can be written in numerous ways and creating a system that can handle all possible code is a demanding and time consuming task. It is also hard or even impossible to develop a system which can identify the connections between different code segments and handle these when new code is added.

Once a function has been added to an application it is a part of the application and can not be removed automatically. If the representation of the function instead was separated from the rest of the application, the source code would still look the same but the XML representation would be different. This would give some interesting benefits. It would be possible to remove the function from the application and more important, updating the function representation would update all the applications using it, which would make maintenance more efficient.

Other enhancements that could make our solution better, would be to extend the amount of source code the application can handle. Making these changes would not be a complicated procedure, but would affect both the parsing and generating of the source code and XML documents.

6 References

Microsoft Visual Studio .NET:
Author: Microsoft
Title: Visual Studio .NET
URL:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsintro7/html/vsstartpage.asp

Microsoft .NET framework:
Author: Microsoft
Title: The .NET Framework
URL:
http://msdn.microsoft.com/library/en-us/vsintro7/html/vxconatourofvisualstudio.asp?frame=true#vxconatourofvisualstudioanchornetplatform

Rational XDE:
Author: Rational
Title: Rational XDE – Extend your development experience
URL:
http://www.rational.com/products/xde/index.jsp

UML:
Author: Object Management Group, Inc
Title: UML Resource Page
URL:
http://www.omg.org/technology/uml/index.htm

XML:
Author: World Wide Web Consortium (W3C)
Title: Extensible Markup Language (XML)
URL:
http://www.w3.org/XML/

CodeDOM:
Author: Microsoft
Title: Generating and Compiling Source Code Dynamically in Multiple Languages
URL:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpcongeneratingcompilingsourcecodedynamicallyinmultiplelanguages.asp

7 Appendix

Appendix 1

A sample of an XML document and the Visual Basic .NET source code produced from it.

Appendix 2

The user manual for our implementation of the solution.

Appendix 1

A sample of

an XML document and

the Visual Basic .NET source code

produced from it.

XML document describing Visual Basic .NET source code:

```
<?xml version="1.0" encoding="utf-8"?>
<Code>
  <Namespace Name="TestExample">
    <References>
      <Reference Name="System" />
    </References>
    <Classes>
      <Class Name="Example">
        <Imports>
          <Import Name="System" />
        </Imports>
        <Attributes>
          <Attribute Name="MyX" Type="System.Int32" Access="Private" />
          <Attribute Name="MyY" Type="System.Int32" Access="Private" />
        </Attributes>
        <Properties>
          <Property Name="X()" Type="System.Int32" Access="Public">
            <GetCode>
              <Line>Return MyX</Line>
            </GetCode>
            <SetCode>
              <Line>MyX = Value</Line>
            </SetCode>
          </Property>
        </Properties>
        <Methods>
          <Method Name="New" Access="Public">
            <Parameters />
            <MethodCode>
              <Line Code="MyY = 5" />
            </MethodCode>
          </Method>
          <Method Name="Add" Access="Public" ReturnType="System.Int32">
            <Parameters>
              <Parameter Name="Z" Type="System.Int32" />
            </Parameters>
            <MethodCode>
              <Line Code="Dim Sum As Integer = MyY + Z" />
              <Line Code="Return MyX + Sum" />
            </MethodCode>
          </Method>
        </Methods>
      </Class>
    </Classes>
  </Namespace>
</Code>
```

The Visual Basic .NET source code corresponding to the XML document above:

```
Imports System

Namespace TestExample

    Public Class Example

        Private MyX As Integer
        Private MyY As Integer

        Public Sub New()
            MyY = 5
        End Sub

        Public Property X() As Integer
            Get
                Return MyX
            End Get
            Set(ByVal Value As Integer)
                MyX = Value
            End Set
        End Property

        Public Function Add(ByVal Z As Integer) As Integer
            Dim Sum As Integer = MyY + Z
            Return MyX + Sum
        End Function

    End Class

End Namespace
```

Appendix 2

User Manual

1 Introduction

This manual is a guide for users developing applications with assistance of the application ProjectManager, an application for automatic code generation into Visual Studio .NET projects, written by Ola Hansson and Anders Svensson during their master thesis. For more information about the application and the theories behind it, we refer to the thesis.

This manual is intended for people with knowledge of Visual Studio .NET. All occurrences where the developer is refer to as a he, this done due to convenience and the developer could of course also be a woman.

2 Installation

The application is run as an add-in to Visual Studio .NET and is using commands in Rational XDE, hence Rational XDE has to be installed with Visual Studio .NET on the computer used for development. To install the add-in in Visual Studio .NET, copy the program library to the desired destination folder on the computer and then follow the instructions available in Visual Studio .NET on how to install an add-in.

In the file ProjectManager.vb the path to the XML files has to be changed to the chosen installation folder. The path is set in the constructor of the ProjectManager class.

```
XMLFiles = ”X:\...\ProjectManager\bin\XMLFiles\”
```

3 Usage

To start the add-in, left click on `ProjectManager` on the `Tools` menu in the Visual Studio .NET as seen in *figure 3.1* below:

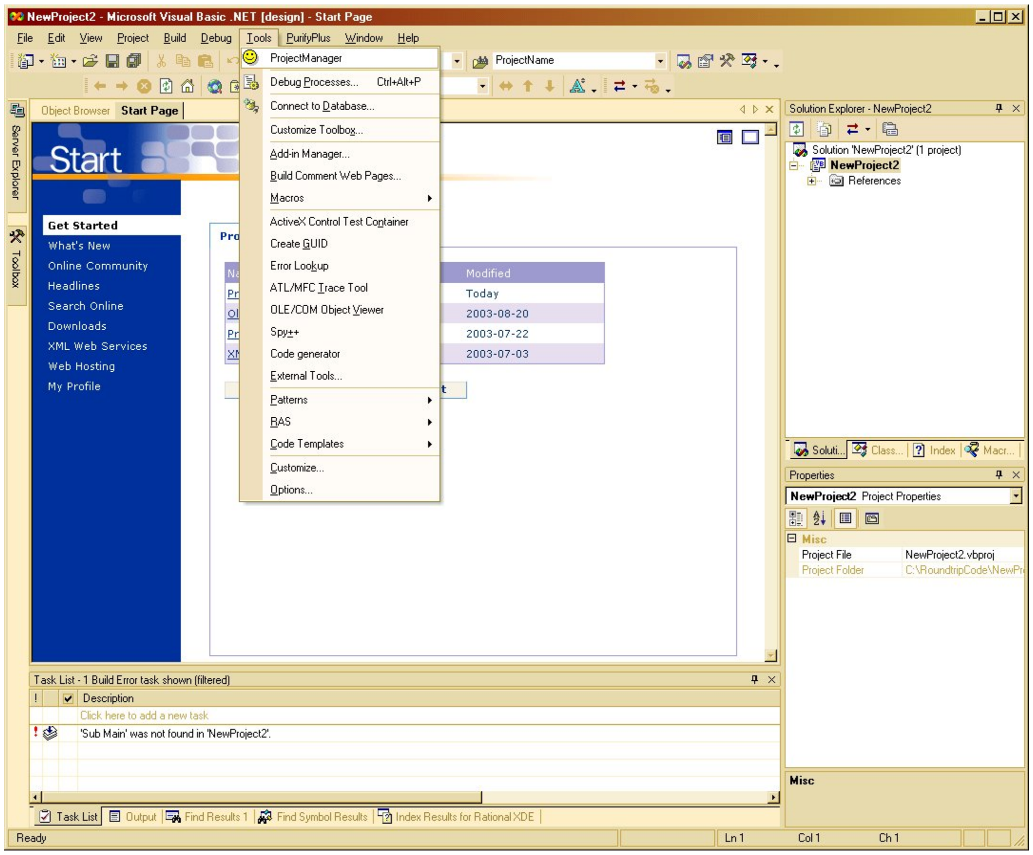


Figure 3.1 Starting the add-in after installation.

At the start of the add-in, a control is being to see if any project is open. If an open project exists, the add-in checks if it has been created with assistance of the `ProjectManager`. This is done by searching for the `generated` folder and checking if there are a system and a log file in this folder. If no project is open the `Create New Project` Dialog pops up, see section 3.1

The basic idea is to work with the add-in in the following way after start up:

<u>If no base system is available:</u>

- Create a new project, section 3.1.
- Create a new base system, section 3.1.1.

<u>If a base system already exists:</u>

- Create a project, section 3.1.
- Open an existing base system, section 3.1.2.
- Add references to the system, section 3.1.3.

Once a base system has been loaded and configured it can be further developed.

The user is now able start to create functions files or add functions from already created function files.

<u>To create a new function to store as a function file:</u>

- Develop a new class which holds the implementation of a new function.
- Save the function to a function file, section 3.2.

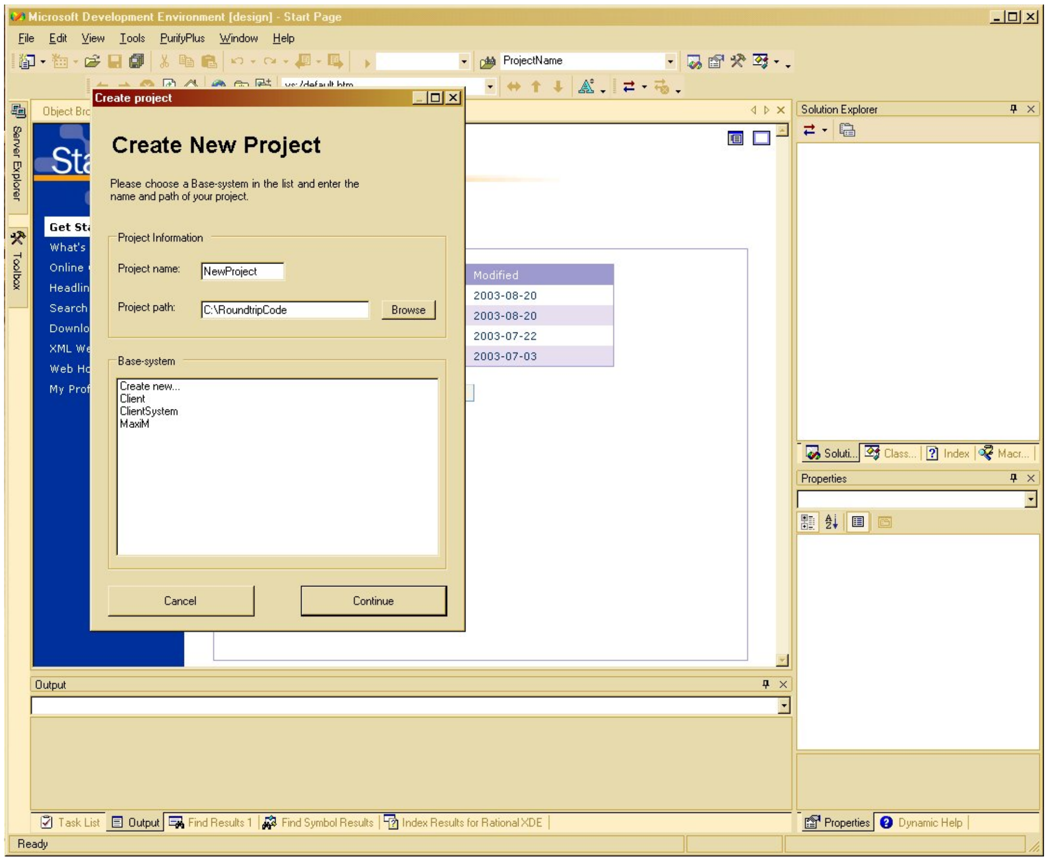
<u>To generate a function from a function file to the current base system:</u>

- Generate functions, from the available function files, to the project, section 3.3. This has to be done one by one i.e. adding one function from a function file at a time.
- Add the references needed by the function, section 3.1.3.

The project is now updated and ready to be further developed.

3.1 Create a new project

If there are no open projects, then the dialog `Create New Project` pops up as seen in *figure 3.2* below:


Microsoft Development Environment [design] - Start Page
File Edit View Tools PurifyPlus Window Help
Create project
Solution Explorer
Properties
Output
Cancel
Continue
Browse
Ready

Create New Project

Please choose a Base-system in the list and enter the name and path of your project.

Project Information

Project name: NewProject

Project path: C:\RoundtripCode

Base-system

Create new...
Client
ClientSystem
MaxiM

Figure 3.2 The Create New Project Dialog.

In this dialog window there are three fields which have to be correctly filled in to be able to create a new project:

- The first field, the `Project name field`, is the name of the project.
- In the next field, the `Project path` field, is the path to the folder where the user wants to store the project.
- In the box at the bottom the `Base-system` box, the user has to choose if he wants to create a new base system or if he wants to use an existing project by mark one of the alternatives.

When all fields have been filled in and chosen the user click the Continue button. If there already is a project with the same `Project path` and `Project name,` the user has to change either the `Project name` or the `Project path`.

3.1.1 Create a new base system

To create a new base system, the user has to choose the alternative `Create new...` from the `Base-system` box in the `Create New Project` dialog, see *figure 3.3* below:

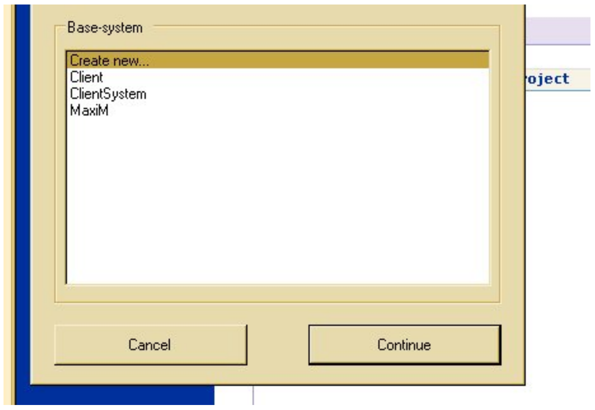
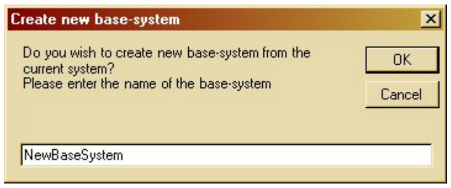


Figure 3.3 The Base-system box from the Create New Project dialog with "Create new…" marked.

After marking the `Create new...`, a new empty project opens when left clicking on the `Continue` button, as seen to right at the bottom of *figure 3.3* above. The user is now able to develop the base system by implementing it. When the user is satisfied with the implementation of the base system, click on the `ProjectManager` in the `Tool` menu in Visual Studio .NET. The dialog shown in *figure 3.4* below pops up and asks the user to enter a name for the base system.


Create new base-system

Do you wish to create new base-system from the current system?
Please enter the name of the base-system

NewBaseSystem

Figure 3.4 The dialog box for giving the base system a name.

By pressing the `Cancel` button the user gets back to the development of a base system without saving it. If the user wants to save the base system, the user has to give the base system a name in the text field in the dialog window and then press the `OK` button. By saving the base system, an XML document corresponding to the base system to create is saved in the folder `XMLFiles` in the `ProjectManager` folder. All the function files that will be generated, will be placed in the subfolder `Functions` in this `XMLFiles` folder.

3.1.2 Open a existing base system

To open an existing base system the user has to choose the base system he wants to work with, by marking the name of that system from the Base-system box in the `Create New Project` dialog as seen in *figure 3.5* below.

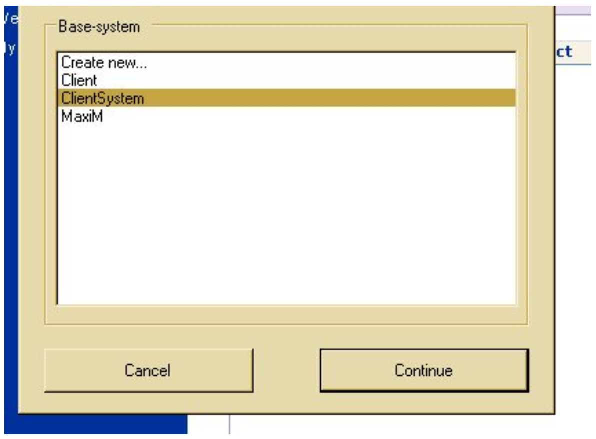
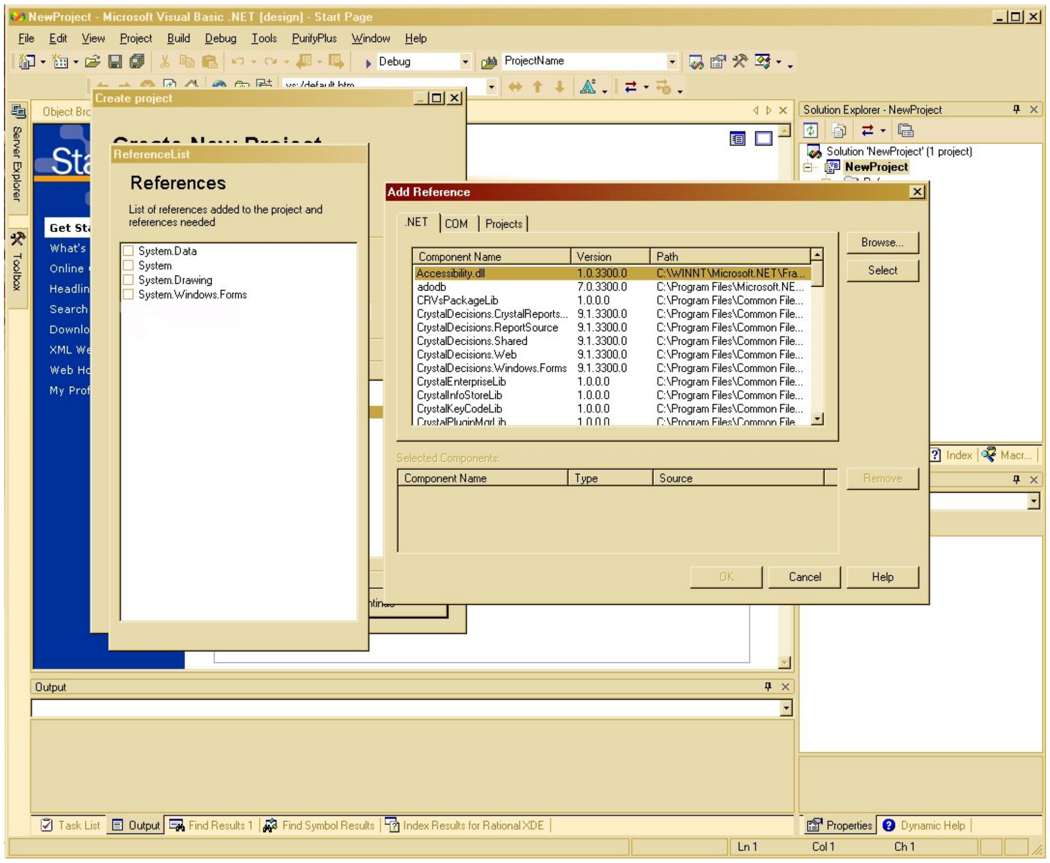


Figure 3.5 The Base-system box from the Create New Project dialog with an existing base system marked.

By clicking the Continue button, the chosen base system is loaded in Visual Studio .NET and further development is possible.

3.1.3 Adding references

The Add reference dialog is used to add the references specified by the XML documents. Two windows are created as shown in *figure 3.6*, one dialog window where the user gets to choose which references to add and one containing a list of the references specified in the XML document. The references in the list must be added, else the code will not work properly.


NewProject - Microsoft Visual Basic .NET [design] - Start Page
File Edit View Project Build Debug Tools PurifyPlus Window Help
Debug
ProjectName
Create project
Create New Project
ReferenceList
References
List of references added to the project and references needed
Add Reference
.NET
COM
Projects
Browse...
Select
Selected Components:
Remove
OK
Cancel
Help
Solution Explorer - NewProject
Solution 'NewProject' (1 project)
NewProject
Output
Task List
Find Results 1
Find Symbol Results
Index Results for Rational XDE
Properties
Dynamic Help
Ln 1
Col 1
Ch 1

- System.Data
- System
- System.Drawing
- System.Windows.Forms

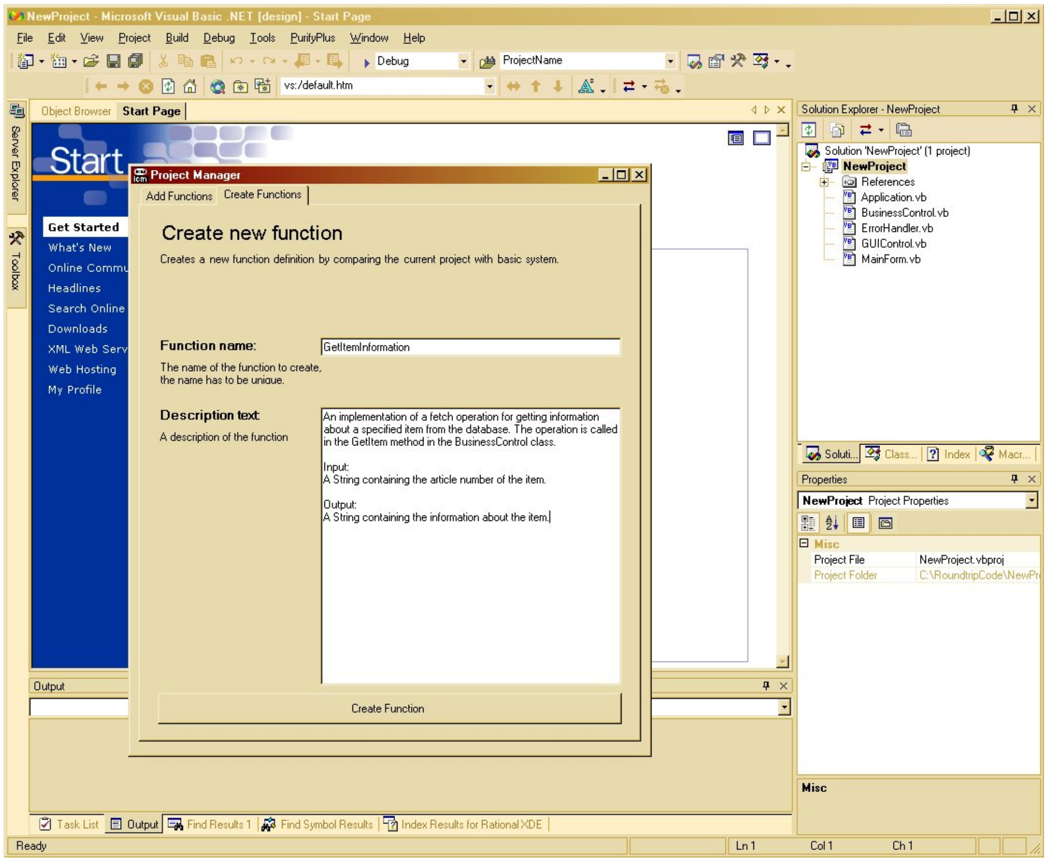
Component Name	Version	Path
Accessibility.dll	1.0.3300.0	C:\WINNT\Microsoft.NET\Fra...
adodb	7.0.3300.0	C:\Program Files\Microsoft.NE...
CRVsPackageLib	1.0.0.0	C:\Program Files\Common File...
CrystalDecisions.CrystalReports...	9.1.3300.0	C:\Program Files\Common File...
CrystalDecisions.ReportSource	9.1.3300.0	C:\Program Files\Common File...
CrystalDecisions.Shared	9.1.3300.0	C:\Program Files\Common File...
CrystalDecisions.Web	9.1.3300.0	C:\Program Files\Common File...
CrystalDecisions.Windows.Forms	9.1.3300.0	C:\Program Files\Common File...
CrystalEnterpriseLib	1.0.0.0	C:\Program Files\Common File...
CrystalInfoStoreLib	1.0.0.0	C:\Program Files\Common File...
CrystalKeyCodeLib	1.0.0.0	C:\Program Files\Common File...
CrystalPluginMgrLib	1.0.0.0	C:\Program Files\Common File...

Component Name	Type	Source

Figure 3.6 The Add Reference windows

3.2 Create new functions

To be able to create function files the user must have developed or opened an existing base system and implemented a new function. To save the function the user has to click on the `ProjectManager` in the `Tool` menu in the Visual Studio .NET and click the `Create Function` tab in the `Project Manager` window that pops up, as shown in *figure 3.7* below:



Create new function

Creates a new function definition by comparing the current project with basic system.

Function name: GetItemInformation

The name of the function to create, the name has to be unique.

Description text

A description of the function

An implementation of a fetch operation for getting information about a specified item from the database. The operation is called in the GetItem method in the BusinessControl class.

Input:
A String containing the article number of the item.

Output:
A String containing the information about the item.

Figure 3.7 The Create functions tab in the Project Manager window.

In this tab there are two fields to enter information;

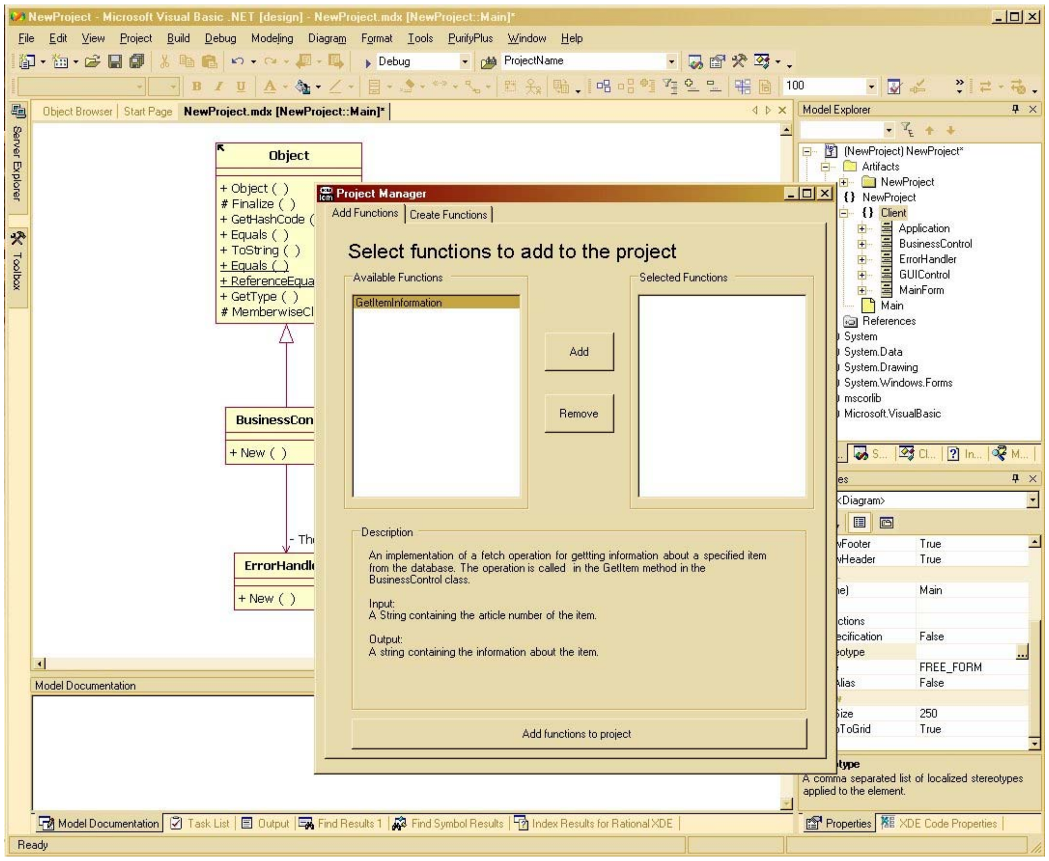
- The first field is the `Function name` field, which gives the user the possibility to choose the name of the function.
- The `Description text` box should contain a description of the function.

It is very important that the description contains all the necessary information about the function.

While developing new function files the user should try to avoid creating conceivable conflicts. This could be avoided by creating method- and attribute names that is unique for that function. For example, use the function name as prefix to all names in the function. Also, the developer should try to avoid using import statements because this can easily create conflicts with other functions.

3.3 Generating a function to the current base system

When the developer has created a couple of functions and stored them, a library with function files exists. These files can be used to decrease the development time by generating these into a current base system. To do this the user has to click on the `ProjectManager` in the `Tool` menu in the Visual Studio .NET and click the `Add Functions` tab in the `Project Manager` window that pops up, as shown in *figure 3.8* below:



An implementation of a fetch operation for gettting information about a specified item from the database. The operation is called in the GetItem method in the BusinessControl class.

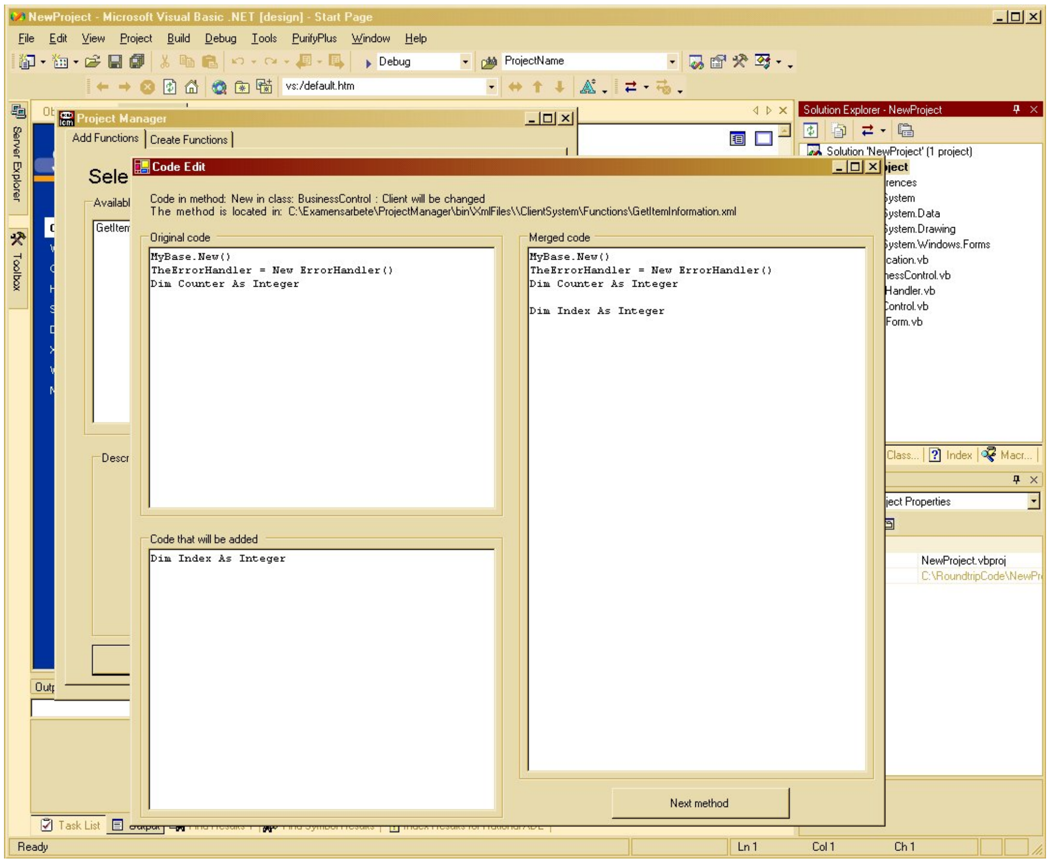
Input:
A String containing the article number of the item.

Output:
A string containing the information about the item.

Figure 3.8 The Add functions tab in the Project Manager window.

In this tab the user can choose which function he/she wants to generate to the base system by clicking on the line corresponding to the desired function. When a function is marked in `the Available Functions` box, the description of the function is shown in the `Description` field at the bottom of the window. To add this function into the base system, the user has to click on the `Add` button between the two boxes. Then the marked function is sent to the `Selected Functions` box on the right. If the user wants to remove an already selected function, mark the function in the `Selected Functions` box and click on the `Remove` button. The generation of the function into the current base system starts when the user presses the button `Add functions to project` at the bottom of the window.

If the code of an existing method in the system will be changed when adding a function, the user is given the possibility to edit the code of the method. This is shown below in *figure 3.9*. The old code of the method is shown to the user in the upper left textbox, below that box the code that is to be added to the method is shown. In the textbox to the right, the user can edit the suggested merge of the code.



Code in method: New in class: BusinessControl : Client will be changed
The method is located in: C:\Examensarbete\ProjectManager\bin\XmlFiles\\ClientSystem\Functions\GetItemInformation.xml

Original code:

```
MyBase.New()
TheErrorHandler = New ErrorHandler()
Dim Counter As Integer
```

Code that will be added:

```
Dim Index As Integer
```

Merged code:

```
MyBase.New()
TheErrorHandler = New ErrorHandler()
Dim Counter As Integer

Dim Index As Integer
```

Figure 3.9 The dialog window for editing merged method code.