Visual Studio Class Designer and Unified Modelling Language

Krisztina Katona

John von Neumann Faculty of Informatics
Budapest Tech
Budapest, Hungary
katona.krisztina@nik.bmf.hu

Abstract: With the launch of Visual Studio 2005, developers got a modelling tool which produces class diagrams from the code or generates code from diagrams. At first sight it hardly varies from UML class diagrams. Besides presenting Class Designer I will show that this difference stems from different software development strategies.

Keywords: software development, class diagram, UML, Visual Studio 2005 Class Designer

1 Visual Representation

1.1 Introduction

There is no software developer who has never used UML diagrams. Unified Modelling Language is taught at universities, discussed in conferences and used by anyone in software industry (it even can be found in text-books about programming like [1]). Class diagrams are the most frequently depicted structures among UML diagrams. Hence the introduction of this widely used tool into a development environment is not astonishing. Although Visual Studio Class Designer builds on UML notation, it did not take from its philosophy in one block.

First I will point out the apparent differences that can be seen on diagrams and then gradually being lost in methodologies. Figures 1 and 2 show class diagrams associated to the same code drawing with UML notation and in Visual Studio Class Designer, respectively. Many slight differences can be recognised at first sight: graphical notations changed in VS, some extra notations occur specialised for a certain language and dependency relationships cannot be depicted. (This example associates to C# code.)

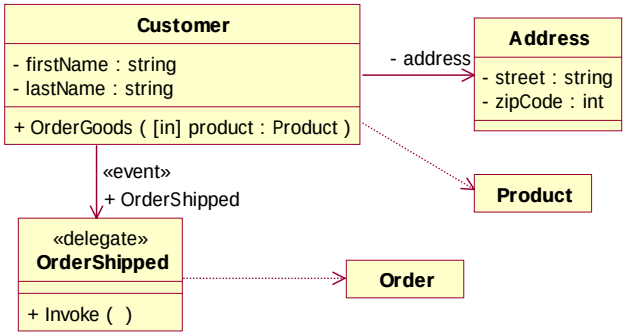

Figure 1

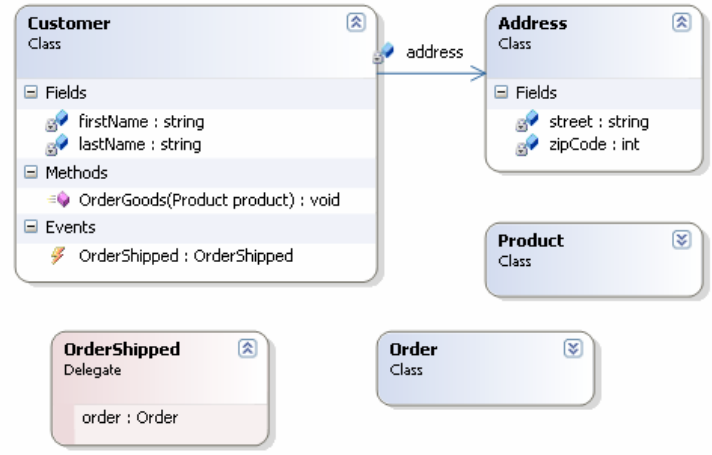

Figure 2

1.2 Differences at First Sight

1.2.1 Layout

Rounded rectangles instead of straight ones in VS express deliberately the difference from UML class notation.

1.2.2 Language-Specific Notation

The language-specific notation is inherently better in VS Class Designer because it is designed for a certain tool namely the Visual Studio, so it derives types from

CLR. 'The Visual Studio Class Designer is a visual code design tool that is an integrated design experience for the .NET Framework. The visual experience of Class Designer is closely tied to the common language runtime. CLR shapes such as classes, structures, and interfaces are represented by visually distinct shapes that indicate their identity. Furthermore, the terminology in the diagram is language specific-for example in Visual Basic, you might work with Public, Private and Friend access levels, whereas in C# they will be displayed as public, private, and internal.' [2] While the UML concept is the opposite: 'UML must work with various implementation languages without incorporating them explicitly.' [3]

1.2.3 Dependency Relationship

The lack of dependency relationships follows from the fact that VS Class Diagram does real-time code generation and generating diagrams from code. The class diagram is actually a live view of the code. A dependency in a model cannot be transformed into code automatically. Hence, it cannot exist in the model. This is the key point where the two concepts vary from each other and it affects deeper distinction. To understand the difference better, observe the Visual Studio's concept thoughtfully.

1.3 Visual Studio 2005 Class Designer

1.3.1 Domain Specific Language

Class Designer is an external component of the Visual Studio supported in Professional Edition and above. It is a Domain Specific Language which means a small, highly focused language for solving some clearly identifiable problem. Microsoft defines Domain Specific Language with the following ideas [4]:

- A model should be a first-class artifact in a project—not just a piece of documentation waiting to become outdated.
- A model represents a set of abstractions that support a developer in a well-defined aspect of development.
- Since models can abstract and aggregate information from a number of artifacts, they can more readily support consistency checks and other forms of analysis.
- Models can be implemented by a process similar to compilation, where the code, configuration files, and other implementation artifacts generated by the compiler are never edited by hand.

1.3.2 File Format

Class Designer features constantly updating with changes between the code and the diagram. Any changes made will be echoed. The class diagram file exists as a part of the project with the extension .cd in plain XML format. The class diagram itself stores only visual information, and contains no information about the content of the code. The cd file of Figure 3 is the following:

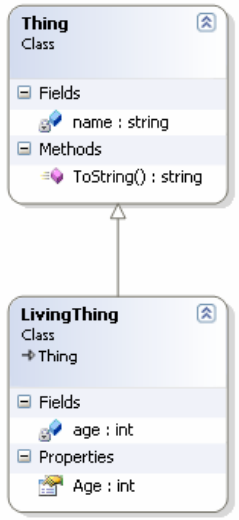

Figure 3

```
<?xml version="1.0" encoding="utf-8"?>
<ClassDiagram MajorVersion="1"
MinorVersion="1"
MembersFormat="NameAndType">
  <Font Name="Tahoma" Size="8.25" />
  <Class Name="ClassLibrarySisy.Thing">
    <Position X="0.5" Y="0.5" Width="1.5" />
    <TypeIdentifier>
      <FileName>BaseClass.cs</FileName>
      <HashCode>AAAAAAAAAAAAAAE-
AAAAAAAAAAAEAAAAAAAAAAAAAAA=
</HashCode>
    </TypeIdentifier>
  </Class>
  <Class Name="ClassLibrarySisy.LivingThing">
    <Position X="0.5" Y="2.5" Width="1.5" />
    <TypeIdentifier>
      <FileName>BaseClass.cs</FileName>
      <HashCode>AAAAAAAAAAAAQAAA-
AEAAAAAAAAAAAAAAAAAAAAAAAA=
</HashCode>
    </TypeIdentifier>
  </Class>
</ClassDiagram>
```

In XML file the ClassDiagram is the top level element which describes the setting applicable to the whole diagram. Then font is determined by the attribute Name and Size. For each class its position and width are described with added information whether the class box is collapsed and about inheritance and association lines. Information about fields and methods of the class are not stored in diagram file.

The fully qualified name of the type associates a shape with the code. While modifying the name of a class, change is reflected in the designer provided that it is open. Otherwise hash code value helps associate the shape to the code. This

value is computed from the member signatures in the class. When the class diagram is opened and does not find the name of a class, the class designer will look for other classes defined in the same file and compute the hash value for those classes. If it finds one within a certain value, then class designer will assume that type to be the one representing the shape and associate it with that shape. [5]

2 Software Development Concepts

The connection between code and diagram files involves that model and source code are essentially equal. This idea leads to the theory that class diagram is the graphical representation of code. Moreover, code is supposed to be the functionality of software. It can be represented many ways such as text or diagram. [6]

2.1 Agile Software Development

The previous conception overlaps and supports agile software development processes. Agile Alliance states four values as manifesto to follow while developing software. These became the basic principles of agile software development: [7]

- Individuals and interactions over process and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following plan

Rapid changes and interactions are preferred to predetermine well-defined plans. Class Designer facilitates this type of work by showing code and model together. As source code and class diagram always reflect each other without any difference the design and implementation phase can be really iterative. It does not take time to switch over from one to the other.

2.2 Unified Modelling Language and Unified Process

On the contrary, UML is a general-purpose visual modelling language which captures decisions and understanding about systems that must be constructed. [8] There are many modelling tools which generate code from UML model and use reverse engineering, but the ease of their usage differs from Visual Studio Class Designer's work. However these tools provide more detailed form of models. They are capable highlight minor differences which can be important in

understanding the model. For example association relationship has two subtypes (aggregation and composition) in UML while they appear in the same way in the code. Hence the distinct between the types of association cannot be derived from the code and therefore Visual Studio Class Designer is unable to visualize the difference.

Unified Process, the methodology built on UML, is a fairly rigid software development method based on spiral model. It clearly defines workflows and makes a distinction between design and implementation phase. Detailed and accurately defined UML models play an important role in the entire development process on all lifecycle stages. [9]

2.3 Different Use

Unified Process and agile development processes stems from the same root, although agile methods can use the experience of UP. Agile developments are suitable for small companies at small or middle projects. At this size quick respond to change can be achieved which is supported by Class Designer. It does not need to analyze the system as deeply as it is important at a large project. Unified Process can cope with large projects by using all aspects of UML. It needs effective proficiency and more abstract skills.

Conclusions

Visual Studio Class Diagram can use the experience of working with CASE tools and UML notations. It is designed for small and middle project which are developed by average skill professionals. Its method is suitable for agile software developments. So Class Designer can be supposed to state at a higher stage of evolution of software development methods. At the same time, large projects need in-depth analysis that can be provided by the whole UML notation. In this case Visual Studio's devices are not enough.

Both concepts contribute to improving of software development quality and hereby better software quality. Different teams need different methods so both concepts have reason for existence side by side. Although I believe that we will see newer and more perfect solutions in software development.

References

[1] Sipos Marianna: Programozás élesben, InfoKit, 2004

[2] Matthew A. Stoecker: Visual Studio 2005 Class Designer, MSDN, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_vstechart/html/clssdsgnr.asp (last visited July 2006)

[3] J. Rumbaugh, I. Jacobson, G. Booch: The Unified Modeling Language Reference Manual, Addison-Wesley, Second Edition, p. 123

[4] Microsoft Corporation: Visual Studio 2005 Team System Modeling Strategy and FAQ, MSDN, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnvs05/html/vstsmodel.asp (last visited July 2006)

[5] ClassDesigner's WebLog: http://blogs.msdn.com/classdesigner/ () last visited July 2006-07-15

[6] R. Hundhausen: Fejlesztői csoportmunka Microsoft Visual Studio 2005 Team System, Szak Kiadó 2006
Original Englis language edition: R. Hundhausen: Working with Microsoft Visual Studio 2005 Team System, Microsoft Corporation, 2005

[7] Agile Manifesto, http://agilemanifesto.org (last visited July 2006)

[8] J. Rumbaugh, I. Jacobson, G. Booch: The Unified Modeling Language Reference Manual, Addison-Wesley, Second Edition, p. 3

[9] J. Tick: Software User Interface Modelling with UML Support, Proceedings of the IEEE 3rd International Conference on Computational Cybernetics, ICCC 2005, Hotel Le Victoria, Mauritius, April 13-16, 2005, pp. 325-328