Introduction

AspectDN (Aspect DotNet) is an Aspect Oriented Programming application targeting software developed with Microsoft's . Net framework. Countless publication and books describe concepts and use cases of the AOP paradigm, I can only encourage you to read them.

AspectDN was originally intended for use in a larger project that I had. I considered therefore developing my own solution because I wanted to go further in the use of this paradigm and I would specify the reasons and how in a dedicated tutorial (tutorial 2).

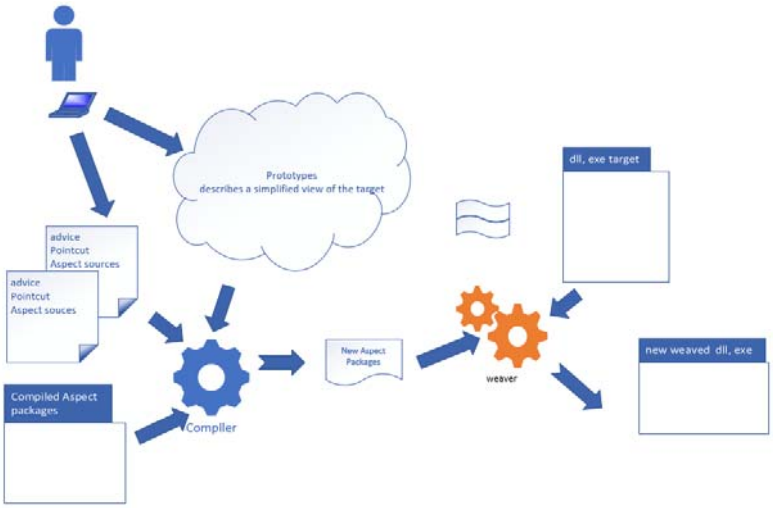
AspectDN relies heavily on Cecil and partly on Roslyn. I also thank the author of the open source ILSpy and Roslyn quoter who helped me a lot in the tests.

What does it do

With AspectDN, you are able to:

- Add class, interface or delegate to an assembly or class of an assembly.
- Add members (field, property, method, event).
- Add inherited classes to an existing class in an assembly.
- Add attributes to members or classes
- Add interfaces to class.
- Add members to interface.
- Add code in different places.

To achieve this, you have at your disposal a language derived from C#5 in order to code aspects, advices and pointcuts. These items are precompiled and these AspectDN precompiled files are used to weave advises in target assemblies at the location you have defined.

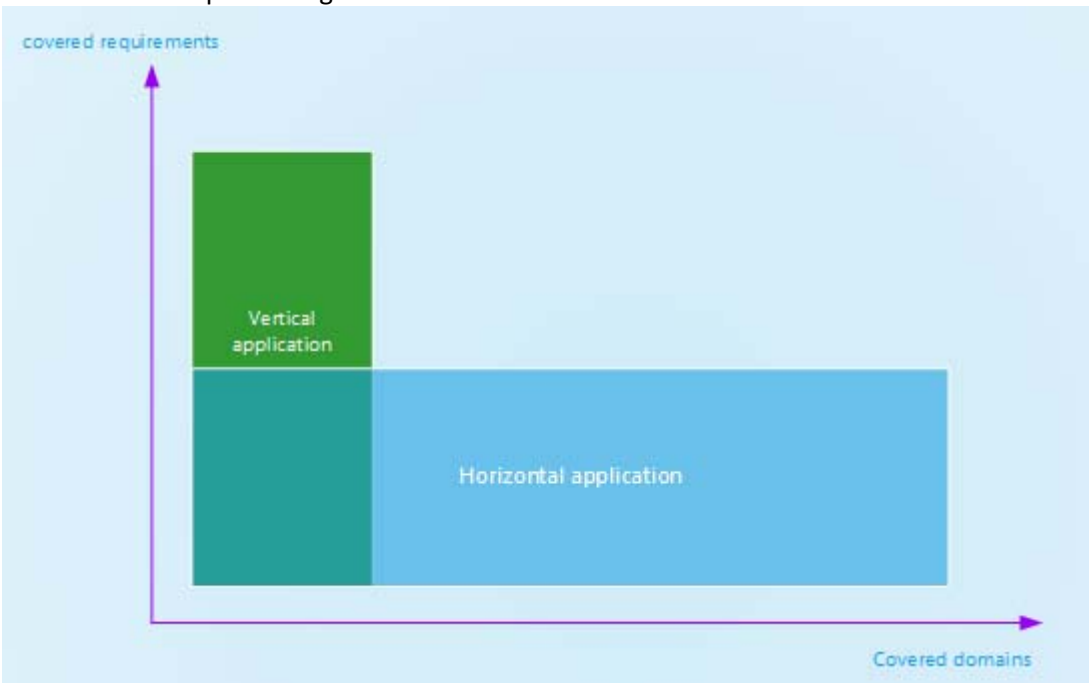


You can be familiarized with by doing the first tutorial.

AOP as a mean to customize software.

The AOP paradigm is not a substitute for object oriented but a complementary means. In all the literature I have been able to find, this paradigm is more or less focused on the technical aspects of an application and less on functional requirements or in the context of software reuse.

Why talk about software reuse?
When you are a software publisher, your application is intended for targets. For example, an ERP supplier has developed its ERP system for specific business domains. The larger the number of domains, the more the software can be sold. An application can schematically be projected on two axes: the vertical axis which identifies the functionalities covered for a defined domain and the horizontal axis representing the different domains that the software could addresses.



A vertical application will adequately cover the requirements of a particular domain and hardly the other areas. A horizontal application will partially cover the requirements of different areas and certainly not as well as a vertical application.

Another constraint with the previous constraint has to be added: the organization of a company. Each company is unique. An application is only a tool and must not impose an organization or processes that are not compatible to an organization. Two companies in a same business are not organized in a same way. You can compare that to a soccer team. Each team has its own way of playing that always depends on the players who compose it with their strengths and weaknesses.

Regarding this, trying to cover all these requirements is not realistic because:

- Some requirements may be contradictory depending on the area
- Trying to fit all requirements into one application will make it too complex to design, implement and test.

The facts we have stated here are also to be taken into account if one is in a multi-tenant Saas configuration where several companies or organizations use the same software but with different requirements in some cases.

But why talking about it here. Simply because AspectDN was designed to be able to customize an application without altering the base application and allowing reuse by:

- By gluing certain components or other external applications with our application.
- Customizing the application according to specific needs (see tutorial 02)