Framework Reuse – Heaven or Hell

JAROSŁAW KUCHTA
Department of Computer Architecture
Gdańsk University of Technology
G. Narutowicza 11/12, 80-233 Gdańsk
POLAND
j.kuchta@eti.pg.gda.pl http://www.eti.pg.gda.pl

Abstract: Object-oriented frameworks have almost completely dominated the world of business applications. Frameworks are widely considered to be helpful and are designed to speed up creation of applications. However, when trying to apply a framework for a particular application, it often turns out that this is not as easy as it seemed at the beginning. It takes time to become familiar with the framework, with its concepts and the standard way of use. Afterwards it often appears that the standard way of using the framework, provided by the authors, do not fully fit for the application. Developers try to match a solution to the existing structure and behavior of the framework. If business requirements to the application are higher than capabilities provided by the framework, The developers try to match the existing framework to save the application. This is what can be called "framework hell". Time is running out, and eventually one may find that the chosen framework should not ever be used in the current application despite the initial similarities to the application requirements. This paper presents a brief analysis of the main problems with the adjustment and reuse of frameworks. The analysis results from three years of the author's experience in framework reuse, particularly with frameworks supporting generation of applications based on Web services in ASP.NET technology. However, the author's experience can be extended to other frameworks, as it results in a number of guidelines to be followed by framework developers to enable other developers to use frameworks in non-standard applications.

Key-Words: framework, reuse, extension, problems, issues

1 Introduction

Application frameworks are build specially to provide developers several benefits [1, 2] such as: maximum of built-in functionality, minimizing risk of developer error, promotion of standard design patterns and code guidelines, extensibility, and portability [3]. However, the effective usage of a framework requires almost expert–level knowledge. As two famous framework experts, Schmidt & Fayad, said: "*Learning to use an OO application framework effectively requires considerable investment of effort. For instance, it often takes 6–12 months to become highly productive with a GUI frame-work like MFC or MacApp, depending on the experience of the developers*" [4]. The most important misunderstanding about frameworks is expectation to spare development time when using them. Moser & Nierstrasz evaluated the effect of object-oriented frameworks on developer productivity and they stated that "*Frameworks do not make developers more productive; they just reduce the amount of work*" [5]. Bosch, Molin, Mattson, and Bengtsson found several problems in framework development, usage, composition and maintenance. One of these problems is "*to understand the intended domain of the framework and its applicability to the application*" [6]. To resolve these problems, emphasis is put on the documentation of the frameworks [7, 8]. However, Kirk, Roper and Wood found out in their experiments with framework documentation [9] that when developers meet problems with framework usage, they search for the problem solution not only in framework documentation (about 60% of attempts), but in their previous knowledge (about 40%). They noticed that when developers tried to fit their previous solutions to the current problem with framework reuse, the time to get the proper solution was longer – sometimes it worked, but mostly did not.

So what opportunities do the developers have when they cannot fit their solutions to the capabilities of the framework? One option is to abandon the framework and to try to use another framework or to write a custom application "from the scratch". There are discussions on this approach within developers community [10, 11]. Another option is to try to fit or expand the current framework capabilities to the requirements of the current application.

At this moment the developers meet a huge bag of problems. Developers forums are full of requests for help in case of framework modification or extension. Their problems can be divided in two categories: understandability problems and inaccessibility of fragments of framework implementation. These two categories of framework reuse problems will be explained in the next two sections of these paper. These problems hit me in my three-year experience as assistant professor in Computer Systems Department of Gdańsk University of Technology – working with my postgraduate students of Web Application Design and cooperating with three industry companies in a project of Comcute – a system for providing big computing power in a crisis situation[1]. My personal experience focused on Microsoft frameworks, such as AJAX, ASP.NET, ADO.NET, WCF, Silverlight, and RIA Services. I am aware that these Windows frameworks are specially hard to extend as in most cases they are provided as dynamic-link libraries (DLL's) – with no source available. I will show some remedies to this obstacle so these "closed" frameworks can be as "open" as frameworks for Linux system. However this does not eliminate the problems with inaccessibility of implementation details. So in conclusions I will provide some guidelines for framework designers. Following these guidelines can cause frameworks to be more flexible and easier to be reused.

2 Understandability problems

Problems with understanding details of framework behavior may be caused by lack of adequate documentation. Kirk, Roper, and Wood identified four categories of such problems: understanding the functionality of components; the interactions between components; the mapping from the problem domain to the framework implementation and understanding the architectural assumptions in the framework design [12]. They studied documentation efficiency for developer problem resolving using three forms of documentation: pattern language, microarchitecture and source code [13]. *Pattern language*, originated from the world of architecture [14, 15], has been adopted to the world o software engineering [16, 17]. It can be simply understood as framework description, mainly hypertext description [18]. The term "microarchitecture" is here meant as a graphical representation of internal framework structure, mainly object and method dependencies [19]. Kirk, Roper and Wood have found out experimentally that the source code has the most important impact on problem solution (more than 50% problem solutions were searched in source code and almost 75% was found). Other authors agree that "*poor documentation can make understandability more complex and a mind–breaking task*" [20].

2.1 Source code as valuable documentation

The source code not only provides an insight into the internal structure of the framework (static relationships), but enables tracing of code execution as well. The tracing capability is a powerful help for the developers in recognition of functional dependencies between currently active objects. I use the term "tracing" instead of "debugging" although the real developer tool is a debugger built in an IDE (Visual Studio for example). However the goal for the developers is to understand the framework behavior, not to remove code errors (at least in the initial period of framework use). The developers set up breakpoints to see where do the execution thread go and at which circumstances (e.g. two breakpoints – one inside and one after an "if" instruction), and watch object changes to see when the object instances are created and what are the results of computation held by object properties.

However, the above mentioned experiment [19] was performed on open-source Java GUI framework JHotDraw[2], with source code provided in a natural manner. What can developers do if the framework is delivered in a form of DLL (without source code)? Let's assume that the framework was designed for Windows system, the most likely for .NET framework, and contains "managed" code. The "managed" code (in contrast to "native" code) means that the code execution is completely managed by operating system (namely by a CLR machine), not by a sole processor itself [21]. Some people say that "managed" code differs from the "native" code in reference counting and garbage collection [22]. That is true, but not the whole truth. "Managed" code is delivered to the CLR machine in a Microsoft Intermediate Language (MSIL) form and it can easily be "decompiled", i.e. translated back to some high–level language as C#, C++, Visual Basic. The main constraint here is that a file with a program symbol database (PDB) is required to be delivered along the DLL file to do so. The PDB file contains a list of program symbols (public, protected and internal) to show "decompiled" code

[1] http://comcute.eti.pg.gda.pl/en/

[2] http://www.jhotdraw.org/

ISBN: 978-960-474-381-0

in a human readable form (with meaningful names of classes and methods).

Some framework authors deliver their frameworks with the PDB files, but some don't. Are the developers hopes buried then? Fortunately, no. Each DLL *must* provide a list of public and protected symbols to be linked to the application. So indeed a PDB file *must* be included in the DLL file. There are some tools known which can "decompile" *almost* every DLL to the source code using these internal PDB's [23]. One of these tools is free Telerik JustDecompile[3] program, which can "decompile" the whole framework and can generate a Visual Studio project that organizes just generated code. Another tools is commercial RedGate .NET Reflector[4] which can be integrated with Visual Studio to enable developer to use internal Visual Studio debugger to get into third–party code and assemblies[5].

2.2 Source code decompilation problems

The word "*almost*" used above is emphasized, because the decompiler results are not fully reliable. It means that the developer *can not* simply get the generated source code to recompile it and to get the exact copy of "decompiled" DLL. There are some obstacles in back–compiling decompiled source code.

First, the DLL *is not* required to provide list of *private* symbols. It is a free decision of a developer to include private symbols in the PDB (and in the DLL) file. If private symbols *are not* included in the PDB, then the decompiler attempts to generate some meaningful names to missing symbols. These names are more or less meaningful and they can be confusing for the developer (see example 1).

Example 1

```
[CompilerGenerated]
internal class <PrivateImplementationDetails>
{A7316BF1-EDFD-42B4-84FD-A64CEF0DCF01}
{
   internal static Dictionary<string, int> $$method0x6000281-1;
   …
}
```

The second issue is that a decompilation algorithm may not be perfect. The MSIL-encoded C# "switch" instruction may be hard to decompile due to unstructured "goto" instructions or due to optimization options used by C# compiler when it compiled C# code to MSIL. I observed "missing label" errors found during compilation of just decompiled code.

The more important problem is a mix of managed and unmanaged code in the same DLL. Basically C# programmers should avoid pointers (known from C and C++) and they should use object types (reference types) instead [24]. The object references are safe, i.e. the managed execution prevents references to uninitialized objects and does not leave *dangling references* to disposed objects. On the other hand, pointers are *unsafe* (and require to use the keyword "unsafe"). It would be perfect if *all* the framework code was written in *safe* (managed) code. However, decompilation of original .NET framework DLL's revealed that Microsoft's programmers were not eager to write "perfect" code. There are many classes and methods marked as "unsafe", especially in mscorlib.dll. When decompiler hits a pointer and pointer operations (e.g. increment, subtraction), it treats each pointer as it would be in "byte*" type. On the other hand, C# compiler takes into account the pointer target size in pointer operations (see example 2).

Example 2

When decompiler hits "char*" pointer moving to the next character, it generates the "+" operator and operand of "2" (in C# each character is two-bytes wide). Next, when this code is compiled back, the pointer hits not "next", but "next after next" character (the proper decompilation should generate "1" as "+' operand value).

```
public override int GetByteCount
          (char[] chars, int index, int count)
  {
    unsafe
    {
      char* chrPointer;

      char[] chrArray = chars;
      char[] chrArray1 = chrArray;
      if (chrArray == null || (int)chrArray1.Length == 0)
        chrPointer = null;
      else
        chrPointer = &chrArray1[0];
      // caution: invalid "2" factor below:
      int byteCount = this.GetByteCount(chrPointer + index * 2,
count, null);
      return byteCount;
    }
  }
```

Mixing of managed objects and pointers requires a special C# "fixed" instruction (which is unsafe). This is because managed objects instances can

[3] http://www.telerik.com/products/decompiler.aspx
[4] http://www.red-gate.com/products/dotnet-development/reflector/
[5] Telerik counterpart for JustDecompile is commercial JustCode

freely be moved in operational memory by execution environment, and pointers require target objects to be firmly set in the memory. "Fixed" instruction is hard to decompile as there are also unsafe instructions which are intended to dispose the dynamically allocated memory avoiding safe garbage collection mechanism of managed execution environment (see example 3).

Example 3

There is a threat in the following decompiled code – although a string parameter is converted to a character array, which is fixed to char* pointer, we access a Length property of the original unfixed string.

```
public override unsafe int GetByteCount(string str)
{
 fixed (char *chars = str.ToCharArray())
 {
  if (chars != null)
  // caution str.Length is unfixed!
    return this.GetByteCount(chars, str.Length, null);
 }
}
```

Often unmanaged code is delivered in an unmanaged DLL. This unmanaged DLL is linked to managed code with a special attribute DllImport. Obviously, unmanaged DLL is impossible to decompile to source code using above mentioned decompilers and a whole bulk of code remains undiscovered.

The most important obstacle is *obfuscation*. Code obfuscation is here a special post-processing stage of compilation from C# (or other managed code language) to MSIL that hides the code intended meaning, which makes the MSIL code hard to decompile [25, 26].

The above problems impede the source code decompilation, and thereby understanding of implementation of the framework. They make impossible to recompile the code generated by the decompiler and make it difficult for developer to trace (debug) the code execution. However, perfectly decompiled code is not necessary for tracing if a decompiler is integrated with IDE (as .NET Reflector is). Integrated debugger allows to trace compiled code and to pass over imperfectly generated source code. On the other hand, valid source code is necessary if the developer needs to modify the framework implementation.

3 Framework code inaccessibility

If the developer can not adjust the solution to the framework capabilities, then the developer have two opportunities to fit the framework to the solution: extension or modification. Extension is the only option if the source code of the framework is unavailable. In such situation the developer may try to write a kind of "framework adapter", i.e. derived classes and methods to change or supplement the functionality. It is the moment when the developer can meet implementation inaccessibility. If the framework is poorly designed (i.e. designed without a perspective to be extended), some data or functionality (e.g. private fields or methods) may be inaccessible. In the following subsection, methods to avoid the inaccessibility obstacles will be shown. However, in my experiences there were also situations, when the developer attempts to reuse the poorly designed (or improperly chosen) framework are condemned to failure.

If the developer attempts to fit the framework to the solution, then tries to write extending or adapting classes derived from the original framework classes. However, this proceeding may be inhibited by some obstacles, such as "sealed" classes and "private" or "internal" class members.

3.1 Sealed classes

First obstacle which hits the developer on the way to framework extension is a "*sealed*" keyword. Classes marked as *sealed* are excluded from the inheritance mechanism, i.e. the developer is prohibited to write classed derived from a sealed class. The "sealed" modifier is intended for optimization – sealed classes do not need virtual method tables to be generated by the compiler, and any virtual method declared in any ancestor class can be invoked directly (not indirectly via VMT)[6]. There are some reports about performance benefits from the sealed classes [27], but these benefits are rather small. Microsoft gurus claims in [28] that: "*You can use this approach to ensure that derived classes do not modify or bypass behavior required for the current class and all derived classes*", but just next to it they warn: "*Do not seal classes without having a good reason to do so. Do not assume that because you do not see a scenario in which extending a class would be desirable, that it is appropriate to seal the class"*.

A limited remedy for the sealed class is to copy source code of the original framework class into an extension framework and to remove the "sealed" keyword. However this solution causes a risk of

[6] The "sealed" modifier can be used also for an overridden virtual method to mark that the current implementation of this method is final and can not overridden in a derived class.

ISBN: 978-960-474-381-0

incompatibility problems when typecasting is used (example 4).

Example 4
Two classes are defined in some framework: a BaseItem class for items and an ItemContainer class for item collection, with an AddItem method. As the BaseItem class is sealed, we cannot derive from this class. So we copy the BaseItem class to a new namespace and remove the "sealed" keyword. Now we can derive from the new BaseItem class and to define a SpecificItem class. However, when we want to add an instance of Specific-Item to original ItemContainer, we get an error, as the SpecificItem class does is not a descendant of BaseItem class.

```
namespace OriginalFramework
{
 public sealed class BaseItem
 { … }

 public class ItemContainer
 {
  public void AddItem (BaseItem item)
  { … }
 }
}

namespace ExtensionFramework
{
 using OriginalFramework;

  // notice: the "sealed" keyword is removed
  public class BaseItem
  { … }

 public class ExtensionItem: BaseItem
 { // in a public constructor we try to add
  // the new instance to the ItemContainer
  public ExtensionItem(ItemContainer owner)
  {
   owner.Add (this); // we get an error here
   // as the ExtensionItem class does not derive
   // from OriginalFramework.BaseItem
  }
 }
```

The same effect of preventing class derivation, but without *any* performance benefit, is declaring a class with a constructor marked with a "*private*" modifier and with no public constructor. Such class is intended to have a static initialization method, and may be used to implement a Singleton design pattern from "Gang of Four" pattern set [29].

If the class in the *is not* sealed, then it may be extended by the developer, i.e. the developer can write a derived class implementing an Adapter design pattern from [29]. Then the developer leaves the current interface of the original framework class untouched, and overrides virtual methods implementation or adds supplementing methods. The benefit is that instances of the adapter class can be used everywhere the instances of the original class may appear. The disadvantage is that the user of the adapter class may incidentally use an original method (if it is not overridden in the adapter) avoiding the adaptor method.

If the class *is* sealed, then the adapter class *is not* derived from this class. However, the adapter class can contain a field referencing to the original, sealed class. Then the developer *must* include declaration of every method required by the application of the newly defined class. The implementation of these methods can be trivial – just to invoke the original class methods – but as it may be many of them, it is a time-consuming task for the developer (example). The next drawback is that instances of the adapter class *can not* be used in place of instances of the original method. Sometimes it forces writing adapter classes to *all* the original classes in the framework classes hierarchy (example 5).

Example 5
Assume that we have a tree of graphical shapes and the shape classes are sealed. If we want to change behavior of one of the shape classes, we need to write an adapter class to this shape class. But as we can not place instances of the adapter class in the shape tree, we are forced to write adapter classes for the whole shape tree.

3.2 Dependent and dependency properties

Model-View-ViewModel design pattern [30] is a kind of the Adapter implementation. Although MVVM is designed to mediate between model (business layer) classes and view (presentation layer) classes, it can be used to wrap any class that implements interface INotifyPropertyChanged [31]. It is a solution for a problem with handling internal property changes. This problem appears when changing one property of the object causes change of another property of the same object. It happens when these two properties are internally bound [32]. A client of this object "will not know" about the change of the second property value until it will get this property. In Microsoft XAML files object properties are often bound to UI components, so that property changes are reflexed in the view changes (using a Binding class [33]). If a business layer class is wrapped by adapter (which is another business layer class), then the chain of notification may be broken. The remedy for it is to generate and handle a PropertyChanged event both in the original framework class and in the adapter class (example).

An alternative for the INotifyPropertyChanged interface implementation is a concept of *dependency properties*. A DependencyProperty is a Microsoft contrivance for declarative programming used in UI-classes [34]. Two UI component properties may be bound in XAML code, so that changing one component property causes change of the other component property change. DependencyProperties are declared as *static* properties and registered for UI-classes. Although they are static, an internal XAML mechanism provides each UI-class instance to have its own collection of dependency properties. They can be accessed via instance (non-static) properties using GetValue and SetValue methods. Registration of a DependencyProperty can be associated with a declaration of a Dependency-PropertyChanged static event handler. Static properties *can not* be overridden (although instance properties can), and their associated property changed event handler can not be overridden also. The remedy for it is to override property metadata by calling the OverrideMetadata method declared in the WPF framework. However the Override-Metadata method is not declared in the Silverlight framework, so it is not an universal remedy. The other remedy requires declaring an instance, virtual counterpart of the DependencyPropertyChanged event handler and to invoke it in the static handler method in the original framework class (example 6).

Example 6

BaseClass, declared as a DependencyObject, registers a static DataProperty dependency property and an instance property Data. The DataProperty change event handling method is a static DataPropertyChanged method. Its instance counterpart is DataChanged method. As it is a virtual method, it can be overridden in an ExtensionClass.

```
namespace OriginalFramework
{
 public class BaseClass: DependencyObject
 {
  public static DependencyProperty DataProperty
   = DependencyProperty.Register
  ("Data", typeof (object), typeof(BaseClass),
    new PropertyMetadata(DataPropertyChanged));

  public object Data
  {
   get { (object)GetValue(DataProperty); }
   set { SetValue(DataProperty, value); }
  }

  public static DataPropertyChanged ( DependencyObject d,
          DependencyPropertyChangedEventArgs e)
  {
   (d as BaseClass).DataChanged(e);
  }

  public virtual DataChanged
          (DependencyPropertyChangedEventArgs e)
  { … }
 }
}

namespace ExtensionFramework
{
 using OriginalFramework;

 public class ExtensionClass: BaseClass
 {
  public override DataChanged
          (DependencyPropertyChangedEventArgs e)
  { … }
 }
}
```

3.3 Private and internal members, non-virtual methods

Even more important problem is caused by members (fields, properties, methods) marked with "*private*" or "*internal*" modifier. In C# internal members are treated as *public* within the same assembly (the same DLL), but as *private* outside of it, so it is the same problem. As object data usually is held in private fields (even the data is delivered to the clients via public properties), the developer has trouble when wants to get access to the original framework class data (example 7).

Example 7

BaseClass declares a DataContext property and holds its value in a private dataContext field. If the dataContext in the current object is not set (has a null value), then the get method of the DataContext property gets the actual value from the owner object of the current object. If we declare an ExtensionClass, we have no chance to know if the data is held by the current object or by its owner. Also the *set* method of the DataContext property invokes a private method RaiseDataContextChanged(). The ExtensionClass has no chance to change the value of the DataContext property without a call to this method!

```
namespace OriginalFramework
{
 public class BaseClass
 {
  private BaseClass owner;
  private object dataContext;
  public object DataContext
  {
   get
   {
    if (dataContext!=null)
     return dataContext;
    else if (owner!=null)
     return owner.DataContext;
```

```
      else
        return null;
    }
    set
    {
     if (dataContext != value)
     {
       dataContext = value;
       RaiseDataContextChanged();
     }
    }
  }

  private void RaiseDataContextChanged()
  { … }
 }
}

namespace ExtensionFramework
{
 using OriginalFramework;

 public class ExtensionClass: BaseClass
 {
  public bool HasDataContext
  {
   get
   {
    // invalid dataContext access attempt
     return dataContext != null;
   }
   set
   {
    // invalid dataContext access attempt
     if (value == false)
       dataContext = null;
   }
  }
 }
}
```

Even when an original framework class declares public (or protected) methods, but these methods call private (or internal methods), then a extension of these public methods may be impossible because of inaccessibility of private methods (example 8).

Example 8

BaseClass declares a private method ValidateData, a protected virtual method SetData and a public method DoSomething. The protected method uses both public and private method. Although in an ExtensionClass we can override the protected SetData method (for instance to call DoSomethingElse), we cannot do this as we cannot use the private method, which is unavailable for the derived class.

```
namespace OriginalFramework
{
 public class BaseClass
 {
  private void ValidateData (object p)
  { … }

  protected virtual void SetData (object p)
  {
    ValidateData (p);
    DoSomething (p);
  }

  public void DoSomething (object p)
  { … }
 }
}

namespace ExtensionFramework
{
 using OriginalFramework;

 public class ExtensionClass: BaseClass
 {
  protected override void SetData (object p)
  {
    ValidateData (p);
    DoSomethingElse (p);
  }

  public void DoSomethingElse (object p)
  { … }
}
```

A remedy for this might be to declare all methods as virtual. On the other hand virtual methods need slots in Virtual Method Tables and their invocations are a bit slower than non-virtual method calls.

Not all problems with virtual methods may be easily solved. If a method is declared as virtual, then when it is overridden in the derived class, it must be declared with the same parameter list and the same result type. We cannot change the result type even if the new would be declared as derived from the original result type. The effect is that the result of the method call must be typecasted.

A remedy for non-public members can be to use a type reflection (using System.Reflection package). This package provides GetMember (and GetField, GetProperty, GetMethod, GetEvent) methods for each class type [35]. These method lets a developer to get info of any class member, not only of a public member. By getting a MethodInfo data, a developer can invoke any method indirectly (see example 9).

Example 9

Like in the example 8, the BaseClass declares a private ValidateData method, a protected SetData method and a public DoSomething method. The protected method uses both the public and private method. Although in the ExtensionClass we can override the protected method, we cannot use the private method that is inaccessible.

However when we get a reference to the private method using a GetMethod, we can invoke it indirectly.

```
namespace OriginalFramework
{
 public class BaseClass
 {
  private bool ValidateData (object p)
  { … }

  protected virtual void SetData (object p)
  {
    if (ValidateData (p))
      DoSomething (p);
  }

  public void DoSomething (object p)
  { … }
 }
}

namespace ExtensionFramework
{
 uses OriginalFramework;

 public class ExtensionClass: BaseClass
 {
   protected override void SetData (object p)
   {
    MethodInfo aMethod = typeof(BaseClass).GetMethod
     ("ValidateData",
      BindingFlags.Instance | BindingFlase.NonPublic);
    if ((bool)aMethod.Invoke(this, new object[] {p}))
      DoSomethingElse (p);
   }
  }

  public void DoSomethingElse (object p)
  { … }
 }
}
```

3.4 Trust levels and friend assemblies

The reflection usage fails in a Silverlight application. These applications are executed in a *sandbox* – a limited framework environment which not only complicates execution of some "risky" operations as file input/output, but also prohibits invocation of non-public methods by the clients [36]. Attempt to use a code from example 9 in a Silverlight application causes a runtime error. Although since Silverlight 2 we could create application in a *full-trust* mode, it was possible only for *out-of-browser* applications. Only in the highest 5th version of Silverlight Microsoft introduced a *elevated-trust* mode for application run inside a browser [37]. This problem is less-weighted as Microsoft slowly retreats Silverlight, however it may be still used in mobile applications [38, 39].

In my experiments with framework code I noticed that internal classes appear widely in Microsoft .NET framework (in mscorlib.dll, system.dll and other libraries). Surprisingly, Microsoft programmers also widely use cross-assembly references to internal classes, i.e. invocations from one assembly to classes defined with "internal" modifier in the other assemblies. It is possible due to a special "InternalsVisibleTo" custom attribute (an analog to "friend" class in C++). We can set names of "friend" assemblies, which are allowed to access to internal classes and class members of the original assembly. However, as these "friend" assemblies have privileged access, Microsoft uses a sophisticated trust mechanism. Assemblies are identified with "strong names" [40], which join "common" names with public keys generated basing on "signature keys". When an assembly attempts to link to internal classes of the current assembly, it checks the strong name of the client assembly (and its signature). Event if we would get full source code of the framework assembly with decompilation and get it reliably recompiled, we could not use internal members of the other framework assemblies because we would get different (invalid) signatures. The only remedy for this is to recompile the *whole* framework (with *all* assemblies), which is extremely difficult.

4 Conclusions

The above analysis brought me to a conclusion that presented remedies are only surrogate solutions to be used when a framework is "poorly" designed, i.e. is designed with no consideration of extension capabilities, and no source code is available. It would be much less of a problem with frameworks reuse if the framework developers obey the following guidelines:

- Deliver source code to the framework, not only compiled code.
- Document your code, not only with internal comments, but with pattern language, examples and microarchitecture explanations.
- Never use *sealed* classes and members.
- Avoid *private* class members, instead consider *protected* or *public* members.
- Never use *private* and *internal* classes and class members.
- Consider wide *virtual* method usage.
- In *managed* framework do not use *unsafe* code. Consider translation from pointers to references.

However the above guidelines are only recommendations for developers. They seem to be

in conflict with the rules of encapsulation in object-oriented paradigm. The first guideline shows clearly the advantages of open-source code over commercial frameworks. We can get rid of a bulk of framework understandability problems. Other authors agree on the benefits of open–source code [41, 42]. On the other hand risks of intellectual property waiver is well known [43, 44]. However I am deep convicted that benefits of wide framework acceptance as a standard will overweight the concerns of framework authors (see [45]).

References

[1] M. E. Fayad, D. C. Schmidt and R. E. Johnson, Building Application Frameworks: Object–Oriented Foundations of Framework, New York, NY, USA: John Wiley & Sons, 1999.

[2] U. Vora and N. L. Sarda, "Framework for evolving systems," in *SEPADS'06 Proceedings of the 5th WSEAS International Conference on Software Engineering, Parallel and Distributed Systems*, Madrid, 2006.

[3] IBM, "Building Object-Oriented Frameworks," [Online]. Available: http://lhcb–comp.web.cern.ch/lhcb–comp/Components/postscript/buildingoo.pdf. [Accessed 2014].

[4] M. E. Fayad and D. C. Schmidt, "Object-Oriented Application Frameworks," *Communications of the ACM*, October 1997.

[5] S. Moser and O. Nierstrasz, "Effect of Object-Oriented Frameworks on Developer Productivity," *Computer*, vol. 29, no. 9, 1996.

[6] J. Bosch, P. Molin, M. Mattsson and P. Bengtsson, "Framework - Problems and Experiences," in *Building Application Frameworks*, M. E. Fayad, D. C. Schmidt and R. E. Johnson, Eds., John Wiley & Sons, 1999.

[7] A. Aguiar, "A Minimalist Approach to Framework Documentation," in *OOPSLA 2000 ACM Conference on Object-Oriented Programiming, Systems, Languages, and Application (Addendum)*, Minneapolis, 2000.

[8] G. Butler and P. Denommee, "Documenting Frameworks," in *Building Application Frameworks*, John Wiley & Sons, 1999.

[9] D. Kirk, M. Roper and M. Wood, "Identifying and Addressing Problems in Object-Oriented Framework Reuse," *Empirical Software Eng.*, 12 2007.

[10] A. Dent, "Unit Testing Frameworks - When to Abandon or Migrate?," 10 October 2008. [Online]. Available: http://www.artima.com/weblogs/viewpost.jsp?thread=240268. [Accessed 2014].

[11] "Should you abandon an ORM framework when you need to implement a bulk operation?," [Online]. Available: http://programmers.stackexchange.com/questions/119003/should-you-abandon-an-orm-framework-when-you-need-to-implement-a-bulk-operation. [Accessed 2014].

[12] D. Kirk, M. Roper and M. Wood, "Defining the Problems of Framework Reuse," in *26th Annual Internation Computer Software and Applications Conference (COMPSAC'02)*, Oxford, 2002.

[13] D. Kirk, M. Roper and M. Wood, "Identifying and Addressing Problems in Framework Reuse," in *IEEE 13th International Workshop on Program Comprehension (IWPC'05)*, St. Louis, MO, 2005.

[14] C. Alexander, "The Origins of Pattern Theory, the Future of the Theory and the Generation of a Living World," in *OOPSLA 1996 ACM Conference on Object-Oriented Programs, Systems, Languages and Applications*, San Jose, California, 1996.

[15] C. Alexander, S. Ishikawa and M. Silverstein, A Pattern Language. Towns, Buildings, Construction, New York: Oxford University Press, 1997.

[16] P. Evitts, A UML Pattern Language (Software Engineering), Sams, 2000.

[17] R. E. Johnson, "Documenting Frameworks using Patterns," in *OOPSLA'92 Conference on Object-Oriented Programming Systems, Lanugages, and Applications*, Vancouver, Canada, 1992.

[18] M. Meusel, K. Czarnecki and W. Köpf, "A Model for Structuring User Documentation of Object-Oriented Frameworks Using Patterns and Hypertext," in *11th European Conference on Object-Oriented Programming (ECOOP'97)*, Jyväskylä, Finland, 1997.

[19] R. Lajoie and R. K. Keller, "Design and Reuse in Object-Oriented Frameworks: Patterns, Contracts, and Motifs in Concert," *Transactions on Pattern Languages of Programming*, 01 2011.

[20] C. J. Satish and T. Raghuveera, "Visualizing object oriented software using virtual worlds," in *SEPADS'05 Proceedings of the 4th WSEAS International Conference on Software Engineering, Parallel & Distributed Systems* , 2005.

[21] Microsoft, "Differences between the Native and Managed APIs," [Online]. Available: http://msdn.microsoft.com/en-us/library/windows/desktop/ee453681%28v=vs.85%29.aspx. [Accessed 2014].

[22] "Difference between native and managed code?," [Online]. Available: http://stackoverflow.com/questions/855756/difference-between-native-and-managed-code. [Accessed 2014].

[23] Z. Maksimovic, "List of Microsoft.NET IL disassemblers," [Online]. Available: http://www.agile-code.com/blog/list-of-microsoft-net-il-disassemblers/. [Accessed 2014].

[24] D. Bolton, "C# Tutorial – About Value Types and Reference Types," 2007. [Online]. Available: http://cplus.about.com/od/learnc/ss/value.htm. [Accessed 2014].

[25] J. Cappaert, *Code Obfuscation Techniques for Software Protection*, Heverlee, Belgium: Katholieke Universiteit Leuven – Faculty of Engineering, 2012.

[26] LogicNP Software, "8 Ways to Protect and Obfuscate Your .NET Code Against Reverse-Engineerign," [Online]. Available: http://www.ssware.com/articles/protect-and-obfuscate-your-dotnet-code-against-reverse-engineering-using-crypto-obfuscator.htm. [Accessed 2014].

[27] "C# Sealed," [Online]. Available: http://www.dotnetperls.com/sealed. [Accessed 2014].

[28] Microsoft, "Limiting Extensibility by Sealing Classes," [Online]. Available: http://msdn.microsoft.com/en-us/library/vstudio/ms229023%28v=vs.100%29.aspx. [Accessed 2014].

[29] E. Gamma, R. Helm, R. Johnson and H. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Professional, 1994.

[30] J. Smith, "WPF Apps With the Model-View-ViewModel Design Patter," [Online]. Available: http://msdn.microsoft.com/en-us/magazine/dd419663.aspx. [Accessed 2014].

[31] J. Liberty, "C#5 – Making INotifyPropertyChanged Easier," [Online]. Available: http://jesseliberty.com/2012/06/28/c-5making-inotifypropertychanged-easier/. [Accessed 2014].

[32] Microsoft, "Implementing the MVVM Pattern," [Online]. Available: http://msdn.microsoft.com/en-us/library/gg405484%28v=pandp.40%29.aspx. [Accessed 2014].

[33] Microsoft, "Binding Class," [Online]. Available: http://msdn.microsoft.com/en-us/library/system.windows.data.binding%28v=vs.110%29.aspx. [Accessed 2014].

[34] Microsoft, "Dependency Properties Overview," [Online]. Available: http://msdn.microsoft.com/en-us/library/cc221408%28v=vs.95%29.aspx. [Accessed 2014].

[35] Microsoft, "Reflection in the .NET Framework," [Online]. Available: http://msdn.microsoft.com/en-us/library/f7ykdhsy%28v=vs.110%29.aspx. [Accessed 2014].

[36] A. Dai, "Security In Silverlight 2," [Online]. Available: http://msdn.microsoft.com/en-us/magazine/cc765416.aspx. [Accessed 2014].

[37] "Silverlight Trusted Applications," [Online]. Available: http://msdn.microsoft.com/en-us/library/ee721083%28v=vs.95%29.aspx. [Accessed 2014].

[38] "Is Silverlight Dead?," [Online]. Available: http://social.msdn.microsoft.com/Forums/silverlight/en-US/f9e15a37-8c5a-4c32-9d38-1d5b57da68de/is-silverlight-dead?forum=silverlightdevtools. [Accessed 2014].

[39] L. Paneque, "Where is Silverlight now?," [Online]. Available: http://www.codeproject.com/Articles/700213/Where-is-Silverlight-now. [Accessed 2014].

[40] Microsoft, "Creating and Using Strong-Names Assembles," [Online]. Available: http://msdn.microsoft.com/en-us/library/xwb8f617%28v=vs.110%29.aspx. [Accessed 2014].

[41] C. G. Carstea, "Open Source ERP," in *Proceedings of the WSEAS international conferences: proceedings of the 1st International Conference on Manufacturing Engineering, Quality and Production Systems (MEQAPS '09)*, Brasov, Romania, 2009.

[42] M. Foltin, P. Fodrek, M. Blaho and J. Murgaš, "Open Source Technologies in Education," in *Resent Researches in Educational Technologies*, WSEAS, 2011.

[43] D. Graeser, "The Benefits and Risks of Open Source Licensing," [Online]. Available: http://www.zdnet.com/news/the-benefits-and-risks-of-open-source-licensing/6354375. [Accessed 2014].

[44] E. J. Walsh and A. J. Tibbetts, "Reassessing the Benefis and Risks of Open Source Software," *Intellectual Property & Technology Law Journal*, January 2010.

[45] B. Perens, "The Emerging Economic Paradigm of Open Source," *FirstMonday*, no. Special Issue #2: Open Source, 2005.