Patterns4Net: Design Patterns Support in Development Environments

Štěpán Šindelář   me@stevesindelar.cz   Faculty of Mathematics and Physics, Charles University in Prague

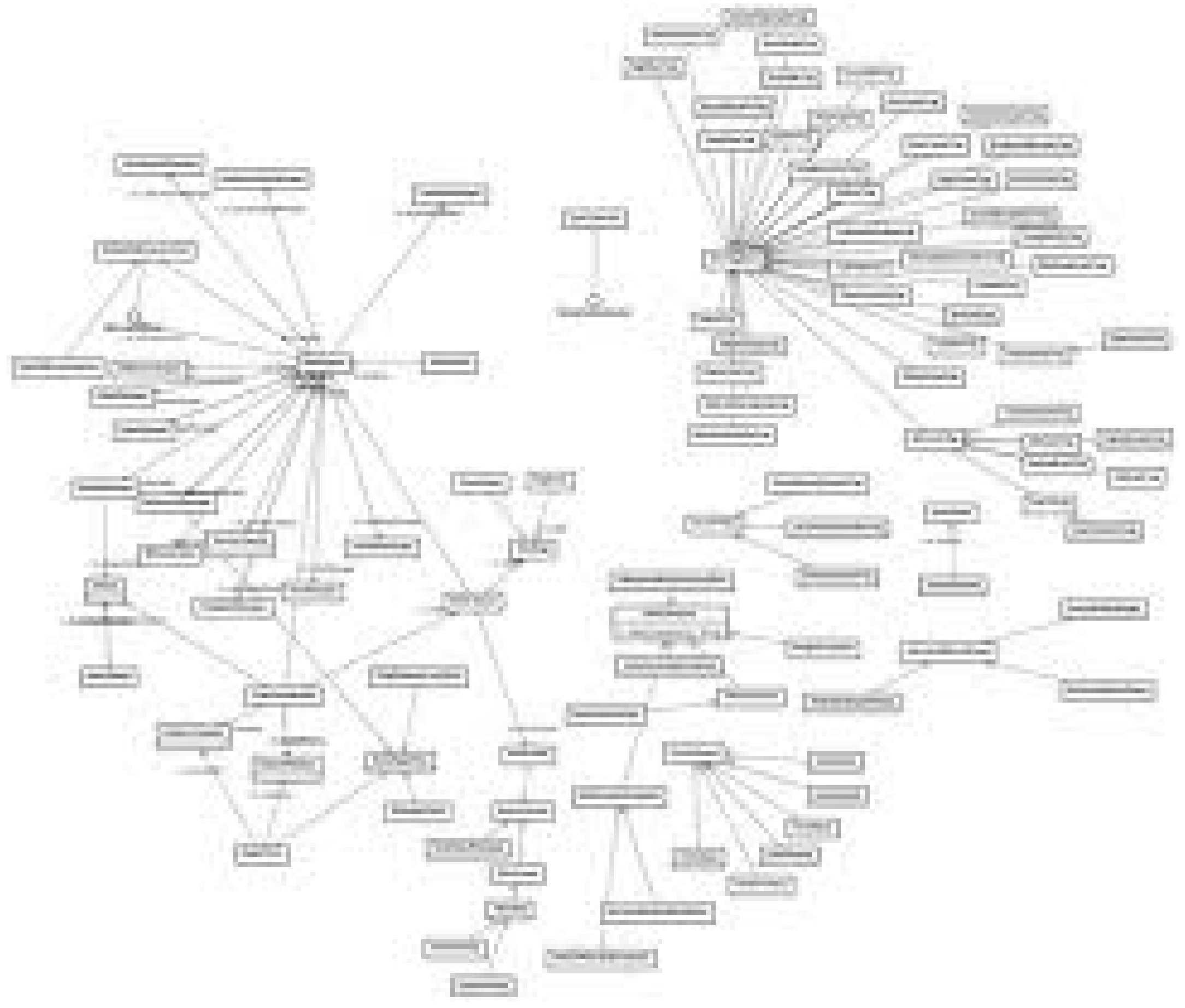
Design Patterns

A design pattern is a description of communicating objects and classes that are customized to solve a general design problem in a particular context. The main aim of patterns in object-oriented design is to **make the design reusable and flexible**. This is very important because frequent changes in the functional requirements are usual nowadays.

Complexity of Design Patterns

Design patterns bring a new complexity into the design, which is **caused by an introduction of new classes and interfaces** in order to provide better flexibility and reusability.

Not Enough Time for Documentation. Developers often don't have enough time to create a textual documentation and so the mapping between classes and a design patterns is lost. Neither the source code, nor reverse-engineered diagrams emphasize the design patterns structure, which would provide more abstract view.



Misunderstanding. Even well documented patterns can be misunderstood, which can slow down the development process or even lead to an introduction of software bugs in the system.

```
/**
 * ...
 *
 * The class is immutable and especially
 * Misho should never ever
 * try to make it mutable :-)
 */
public class SchedulerInfo ...
```

Existing Approaches

Formal Verification. There are several approaches for formal verification of design patterns implementation. Most of them target the Java platform or C++, but we are not aware of such tool for the .NET platform. Moreover, too much mathematical formalism is usually involved in their usage, which makes them harder to be used by average developers.

Reverse Engineering. Tools for tackling the complexity of design patterns are mostly based on an automatic recognition of design patterns, whose advantage is that it does not require additional work from developers and can be used for legacy systems, but it's disadvantage is that it cannot correctly recognize all the design patterns, since the differences between some of them are only semantical (the *Bridge* and the *Adapter* patterns) and some patterns, such as the *Command* pattern, are too much abstract to be recognized only from the source code.

Patterns4Net

The main conception behind the Patterns4Net is that developers will annotate their code using .NET attributes mechanism and the Patterns4Net will provide tools that will take advantage of this documentation and will support the development process of design patterns oriented software.

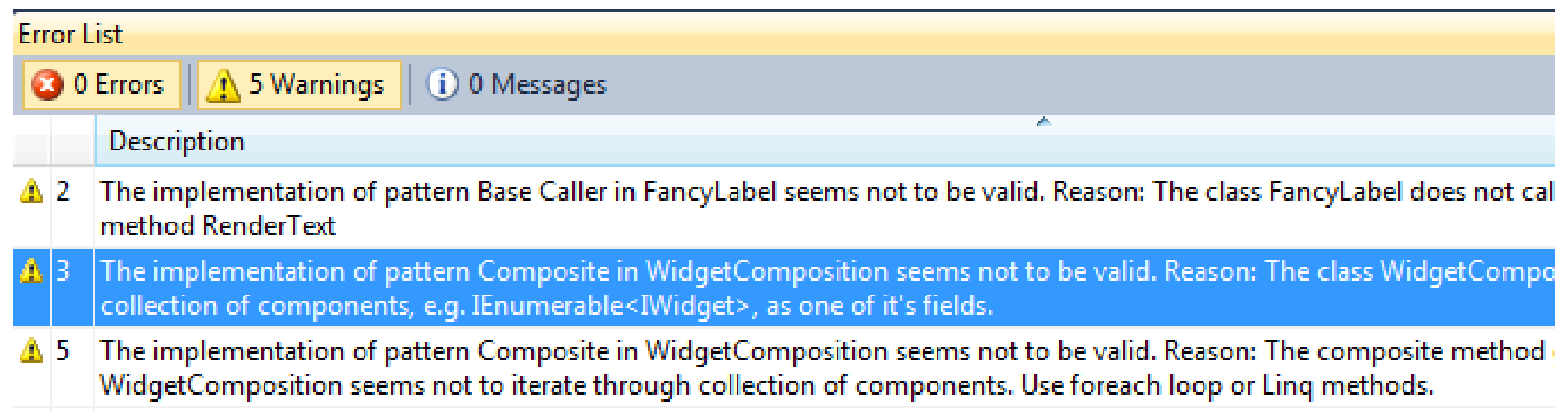
```
using Patterns4Net.Attributes;

[Composite(typeof(IWidget))]
public class WidgetComposition : IWidget, ICloneable ...

[Visitor(typeof(IWidget))]
public abstract class WidgetsVisitor ...

[Immutable]
public class Position ...
```

Patterns4Net provides two tools:

- **Pattern Enforcer** verifies some of the structural aspects of selected design patterns implementation.
- **Architecture Explorer** generates interactive UML-like class diagrams from .NET assemblies. This tool uses the information about design patterns implementations to generate more abstract and high-level diagrams than standard UML reverse engineering tools.

Pattern Enforcer

Pattern Enforcer is a tool that verifies correctness of selected structural aspects of design patterns implementations. For example, it verifies that all elements that can be visited by the *Visitor* indeed override and correctly implement the *Accept* method, which is essential for the correct implementation of this pattern, but unfortunately not enforced by the standard compiler. There are 14 built-in patterns, but users can also add their custom patterns or even simple idioms using the special type-safe domain specific language embedded into the C# language. We believe this approach is easier to use than most of the other formalization techniques that use special languages.



Error List

0 Errors | 5 Warnings | 0 Messages

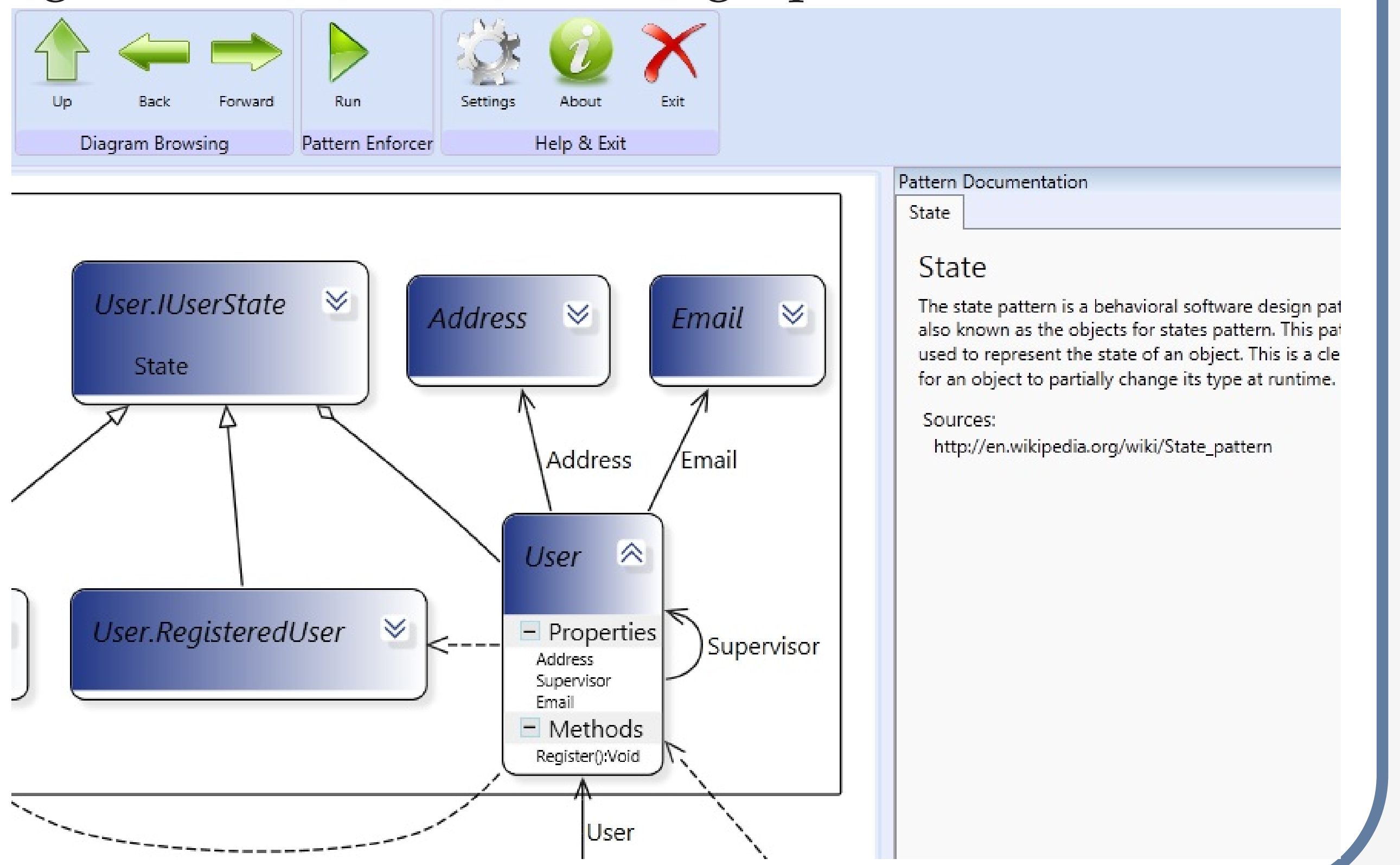
		Description
⚠	2	The implementation of pattern Base Caller in FancyLabel seems not to be valid. Reason: The class FancyLabel does not cal method RenderText
⚠	3	The implementation of pattern Composite in WidgetComposition seems not to be valid. Reason: The class WidgetCompo collection of components, e.g. IEnumerable<IWidget>, as one of it's fields.
⚠	5	The implementation of pattern Composite in WidgetComposition seems not to be valid. Reason: The composite method WidgetComposition seems not to iterate through collection of components. Use foreach loop or Linq methods.

Pattern Enforcer can be integrated into the Visual Studio build process or used as a strand-alone command line program.

Implementation. Patterns4Net and Pattern Enforcer analyzes CIL (common intermediate language) code in .NET assemblies, which means it can be used for any programming language that can be compiled into the CIL.

Architecture Explorer

Architecture Explorer provides UML-like class diagrams generated from .NET assemblies. Instead of a one large diagram with lots of unnecessary infrastructural classes, it uses the information about implemented design patterns to create more diagrams with different levels of abstraction. For example, the top level view shows only the high level, domain specific classes that are important for understanding of the overall architecture. However, if a user "zooms" to a particular class, all related classes, even infrastructural, are displayed. Which elements are displayed and which are not, is chosen according to design patterns they implement. For instance, the *NullObject* pattern represents rather implementation detail. Users can browse the diagrams in an interactive graphical user interface.



Pattern Documentation

State

State

The state pattern is a behavioral software design pat also known as the objects for states pattern. This pat used to represent the state of an object. This is a cle for an object to partially change its type at runtime.

Sources:
http://en.wikipedia.org/wiki/State_pattern

Related Work

The main source of inspiration for Pattern Enforcer was Pattern Enforcing Compiler for Java (PEC). It is extended Java compiler that verifies design patterns. For patterns annotation PEC uses marker interfaces. This technique has few drawback (e.g., methods cannot be annotated with interfaces), which the authors of PEC have admitted and they planned to support Java annotations (similar to .NET attributes used in our Enforcer) in the next version. We are, however, not aware of any updated version of PEC with Java annotations. PEC also does not provide any special API or language for custom patterns specification as we do.

Conclusion

Patterns4Net can enhance the development process of complex design patterns oriented systems created by larger teams, because it helps to discover communication errors and violations of design patterns implementations earlier and it provides visual tool to tackle the design complexity that is caused by design patterns usage.

We are not aware of any other design patterns verification tool for the .NET platform. Pattern Enforcer is, among all of these tools, also extraordinary with it's special C# API for structural constraints specification, because most of the other approaches use special languages for this purpose.