

# NShape - An open source diagramming framework for .NET WinForms

---

## White Paper

Author: Peter Pohmann, peter.pohmann@dataweb.de

Copyright by dataweb GmbH, Aicha, Germany

Date: 2009-09-20

Version: 1

Diagramming frameworks are complex application parts, which allow the user to view, annotate, modify and create diagrams consisting of connected shapes. There exist currently some diagramming frameworks for .NET on the market, but their primary purpose seem to be small and simple diagrams. Industrial applications have requirements that go far beyond the capabilities of these existing frameworks.

## 1 Industrial Application Requirements

dataweb has analyzed a set of industrial applications and their need for diagramming capabilities. The sectors included railway, energy, automation and manufacturing. The diagrams included flow charts, wiring plans, railway networks, plants, logical functions and others. We have found six major requirements for these industrial applications.

### 1.1 Large Diagrams

Industrial diagrams often have 10,000 shapes or more. This is not only a challenge for the performance of the diagramming framework but also for the user experience concepts. When a user has to work with 100 diagrams, each consisting of 10,000 shapes he must dispose of means to search and modify within these one million shapes in a consistent way in reasonable time.

### 1.2 Separation of Presentation and Model

Within industrial applications, diagrams are much more than just a presentation means. The entities which are visualized through the shapes in the diagram have usually a complex behavior, which is very specific to the given application. The diagramming framework must provide a way to connect the framework's presentation objects – the shapes - to the specific semantics of the application domain. For example, if an application is about transformers, the data and behavior of a transformer must be tied to the transformer shape.

### 1.3 Persistency Integration

Industrial applications use various persistency mechanisms to store their data on disk. In most cases it is of critical importance that this data be manageable and consistent. Applications must be capable of ensuring and checking this consistency. Therefore, a diagramming framework must be able to integrate its own persistency with the application's persistency.

If for example the diagramming framework stores its diagrams in XML files completely separated from the rest of the application data, which might reside in an RDBMS, consistency can be neither guaranteed nor checked.

## 1.4 User Interface Integration

Most industrial application's user interfaces differ from common business applications. They use non-standard user interface controls on a regular basis and often employ more sophisticated patterns for binding the controls to presentation logic and model data. A diagramming framework must be able to fit into such a user interface approach and adopt the application's general look & feel.

## 1.5 Diagram Dynamics

The entities in industrial applications often contain dynamic states, for example switches can open and close, logical functions evaluate depending on their input signals, trains move in the network, robots take and deposit workparts etc. Diagrams must be able to reflect this dynamic in the presentation, be it the color, size, pattern or position of a shape.

## 1.6 Extensibility

Industrial application use a disparate set of shapes much more varying than business applications. Therefore, it is indispensable for a diagramming framework to provide means for adding application specific shapes. But shapes alone are not enough, almost every aspect of the diagramming frame work must be extensible: Persistency, user interface, tools etc.

# 2 How NShape Satisfies Industrial Requirements

dataweb has invested thousands of hours to find technical solutions to the specific diagramming requirements of our industry clients.

## 2.1 Large diagrams

In order to cope with large diagrams an extensive set of technical and design measures were taken.

### 2.1.1 Template-Based Shapes

In NShape, all shapes are by default based on the template, they were originally created from. Many properties of the shape are bound to the template and adjust when they are modified there. For example, when the user modifies the color of the template, all shapes deriving from the template change their color accordingly. The user can separate the shape's property from the template by expressively assigning it its own value.

### 2.1.2 Design-Based Shapes

The shapes in NShape are also design-based. When a shape is for instance red, there is no actual color assigned to the shape. Instead, its color is bound to a style with the name "red". Now, when the user modifies the red style to a darker or lighter red (or even to blue, but that's not recommended), all shapes with that style adjust themselves immediately. The same thing works for line styles, fonts and other design elements. That way one can alter the general design of the diagrams consistently in a very comfortable way.

### 2.1.3 SQL Repository

For large diagrams or large numbers of diagrams, a file based persistency like XML files are no more adequate. Therefore NShape also comes with an option to store

diagrams in SQL databases. The repositories in general are designed for partial loading and saving to get high performance even with big diagrams.

#### 2.1.4 Smart Implementation

All algorithms within NShape are designed with large diagrams in minds. For example, shape lists are stored within a spatial access structure.

## 2.2 Separation of Presentation and Model

Modern applications often contain an object-oriented model of the problem domain. The model objects then represent the information and behavior of the real-life objects within the application. When these objects are to be visualized within a diagram, it makes sense to bind the representing shapes to the domain objects they represent. This way, the business logic of the domain object determines the appearance and behavior of the shapes in the user interface.

Let us look at a battery as an example. Its behavior in the real-world is to have two poles and to emit electric current for some finite time. Therefore its screen counterpart should not allow connecting more than two wires and indicating its load state by a fading color in correspondence to the actual charge.

Since this business logic is already present in the application, it would be a bad idea to duplicate this behavior into some specialized shape object or shape scripts. Instead, NShape defines an interface through which the framework can connect its shapes directly to the already existing domain object, the *IModelObject* interface. This way maximum consistency, integrity and code reuse can be obtained.

## 2.3 Persistency Integration

The root of NShape's persistency support is the *IRepository* interface together with a set of helper interfaces and classes. Each NShape class is capable of serializing and deserializing itself through this interface. NShape also contains two implementations of this interface, one for XML files and the other one for SQL Server.

This design allows for a variety of persistency scenarios, for examples these ones:

- a) An application without domain model layer uses the out-of-the-box XML repository or SQL Server repository.
- b) An application with domain model layer implements the NShape persistency mechanism for its model objects and serializes all its data through *IRepository*.
- c) An application with or without domain model layer implements its own repository (e.g. for Oracle based on the provided *AdoNetStore* base class) and serializes all its data through this interface.
- d) An application serializes only the NShape classes through *IRepository* and its own classes through its own persistency mechanism. Nevertheless, all data can be physically stored in the same file or database schema.

## 2.4 User Interface Integration

All NShape user interface elements follow the controller-presenter pattern to localize as much of the presentation logic as possible outside the actual Windows control. It is therefore very easy to exchange the controls. A company which uses WPF controls or control libraries from some third party vendor can use them also for NShape. This way seamless integration of the user interface is made possible, without having to duplicate or re-implement the presentation logic.

## 2.5 Diagram Dynamics

In order to bind the shape's visual properties to the model object's state properties, mappings can be defined between them on the template level. Together with the *StateChanged* event in the *IModelObject* interface, real-time presentation of dynamic model states can be realized with very little additional effort in the application itself.

## 2.6 Extensibility

In order to provide extensibility, NShape is interface based and comes with a large set of base classes with many public and protected members. NShape uses the dependency injection principle to integrate model objects and proprietary shapes into the framework.

## 3 Additional Information on NShape

NShape has a dual license from which the NShape user chooses the one, which fits his needs most:

- a) GNU General Public License v3 for free software, or
- b) Commercial license for proprietary software.

NShape v1 is currently in public beta. Everybody can download the current version and evaluate it for non-commercial projects.

For further information on NShape, please turn to

- the Open Source homepage on Google Code: <http://code.google.com/p/nshape/>, or
- the dataweb web site: <http://www.dataweb.de>, or
- dataweb directly: [sales@dataweb.de](mailto:sales@dataweb.de)

Oracle is a trademark of Oracle Corporation.  
SQL Server is a trademark of Microsoft Corporation.