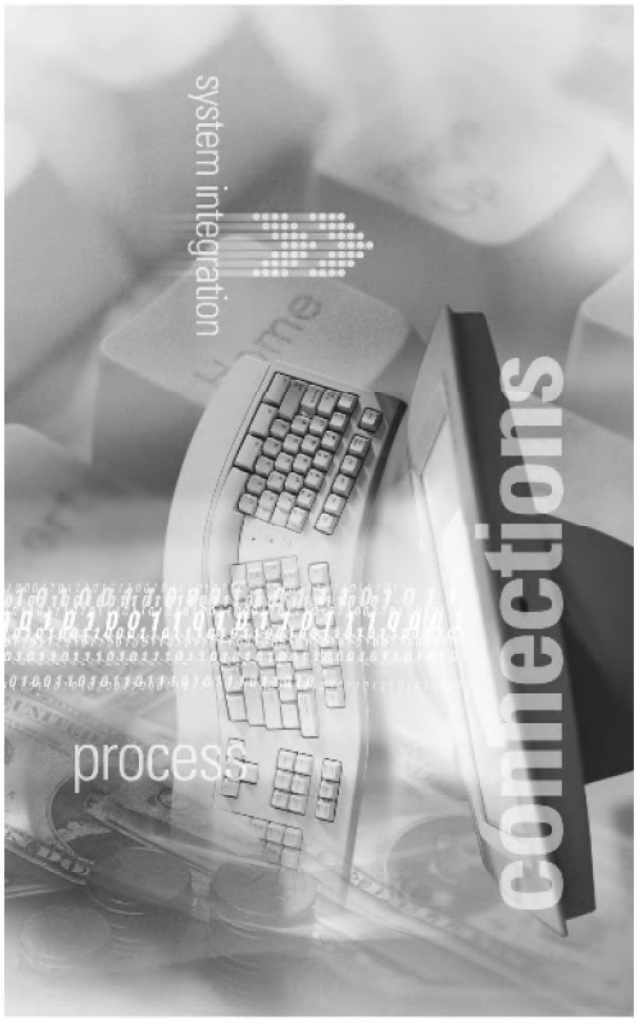
NuvoControl

Software Architektur und Design



Produkt:	**NuvoControl** V1.0
Doc-Nr:	NuvoControl_1300
Dateiname:	NuvoControl_1300_Software_Architektur_und_Design.doc
Verantwortlich:	B. Limacher, Ch. Imfeld
Ausgabe:	g / 17-Aug-2009
Datum:	8/17/2009 6:30:00 PM

Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patenterteilung oder Gebrauchsmuster-Eintragung vorbehalten.
Copyright reserved

Limacher Bernhard
Imfeld Christian

ABBILDUNGSVERZEICHNIS

1. GENERAL

1.1 Änderungen gegenüber der Vorgängerausgabe

This document has the version: **g / 17-Aug-2009**

Index	Datum	Verantwortlich	Änderungen
a	29.04.2009	B. Limacher, Ch. Imfeld	Erste Ausgabe
b	24.05.2009	B. Limacher	Kapitel Architektur hinzugefügt.
c	28.06.2009	B. Limacher	- Chapter: Architecture reworked - Various diagrams and short descriptions of service components added.
d	29.06.2009	B. Limacher	- Server startup sequence added. - Description of communication patterns added - Description of hosting added.
e	15.08.2009	B. Limacher	- Description of the client added. - Chapters 3 till 5 updated
f	15.8.2009	Ch. Imfeld	Add description of the protocol driver. Beautify document.
g	17.8.2009	Ch. Imfeld	Add description of the NuvoControl simulator.

1.2 Zweck

Dieses Dokument beschreibt die Software Architektur und das Design von NuvoControl.

1.3 Anwendungs- / Geltungsbereich

Dieses Dokument gilt für das NuvoControl Projekt.

1.4 Referenzierte Dokumente

Ref.	Dokument-Nr	Titel
[1]	NuvoControl_0103	Projekt Glossary, enthält Projektweite Abkürzungen (zusätzlich zu den im nächsten Abschnitt aufgeführten Abkürzungen spezifisch für dieses Dokument)
[2]	http://www.nuvotechnologies.com/	Homepage of Nuvo Technologies

1.5 Abkürzungen + Konventionen

Liste von Abkürzungen spezifisch für dieses Dokument. Die globalen Projektweiten Definitionen sind in [1] aufgeführt.

Abkürzung	Beschreibung
DAL	Data Access Layer
NuvoEssentia	Nuvo Essentia Multi Room Verstärker

Unvollständige oder noch zu vervollständigende Teile dieses Dokumentes sind in Spitzklammern '<' . . . '>' gekennzeichnet. Die Zeichenfolge'**TBD**' (to be defined) wird dabei zu Beginn der Spitzklammern gesetzt.

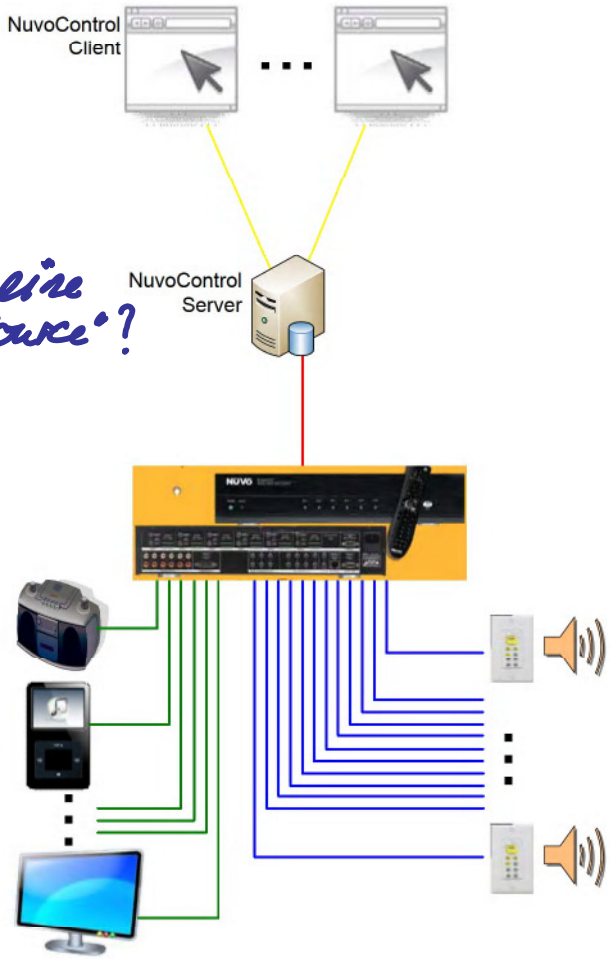
aber < > nicht verwendet

2. Overview

Sehr gut

Bei dem Projekt NuvoControl handelt es sich um eine Software zu Ansteuerung und Visualisierung einer Audio Multiraum Anlage von Nuvo Systems (siehe [2]).

Das Bild zeigt eine Übersicht der NuvoControl Software mit der zu steuernden Multiraum Anlage.



Das Multiraum System hat zwölf Zonen. Jede Zone verfügt über ein Keypad zur Steuerung der dazugehörigen Zone und zwei Lautsprechern. Das Keypad erlaubt mit seinen Tasten die direkte Wahl der gewünschten Audio-Source (im Bild blau eingezeichnet).
Das Nuvo System erlaubt die Integration von bis zu sechs beliebigen HiFi Geräten (diese Verbindung ist grün eingezeichnet).
Zusätzlich besitzt der Verstärker eine RS232 Schnittstelle, über die die Multiraum Anlage gesteuert werden kann. Es können sämtliche Funktionen ausgeführt sowie alle Zustände der Zonen abgefragt werden (diese Verbindung ist rot eingezeichnet).

Ist das wirklich eine Source?

Die Visualisierungs- und Steuersoftware **NuvoControl** der Audio-Multiraum-Anlage steuert die Multiraum Anlage über die RS232 Schnittstelle und visualisiert den Zustand des ganzen Systems. Die Software erweitert die existierenden Möglichkeiten der Anlage um weitere Funktionalität, wie Sleep- und Weckfunktion.

Die Software NuvoControl basiert auf einer Client-Server Architektur, welche mehrere Clients unterstützt.

- Erste Kernkomponente ist der **Monitor & Control - Service** welcher die Steuerung von NuvoEssentia erlaubt sowie den Zustand von NuvoEssentia abfragt und für eine Visualisierung zu Verfügung stellt.
- Zweite Kernkomponente ist ein **GUI**, welches den Zustand von NuvoEssentia graphisch darstellt und Bedienoperationen unterstützt.
- Dritte Kernkomponente ist der NuvoEssentia **Treiber**, welcher das eigentliche Protokoll zu NuvoEssentia implementiert.
- Vierte Kernkomponente ist der **Function - Service**, welcher die Sleep- und Weckfunktion implementiert.
- Die gesamte Software ist mittels XML Files konfigurierbar. Anzahl Zonen, Adressierung der Zonen, Visualisierung der Zonen, verwendete Kommunikationstreiber und weitere Einstellungen sind dynamisch änderbar.
- Sämtliche Softwareteile sind eventbasiert (Subscription Pattern).

Für die Projektrealisierung wurden haben wir die folgenden **Technologien** und **Methoden** eingesetzt:

- Implementation mit C#
- Client-Server Kommunikation mit Web Services, implementiert mit WCF
- Server zu Multiraum-Simulator Kommunikation mit Queues (MSMQ)
- GUI Implementation mit Windows Presentation Foundation (WPF)
- Visual Studio 2008 mit dem Unit Test Framework von Microsoft (VSTS)
- Continuous-Build Integration mit CruiseControl.NET
- Source Code Verwaltung mit Subversion und Tortoise
- Dokumentation erstellt mit Hilfe von Microsoft Sandcastle (NDoc) und Doxygen
- UML Modellierung mittels Enterprise Architect (EA)
- Use-Case basiertes Requirements Engineering (nach Cockburn / Bianchi)
- Requirements Driven Development (RDD) (gemäss Wirfs-Brock / Tobler)
- Agiler Entwicklungsprozess gemäss SCRUM

3. Architecture / Design Overview

The architecture of NuvoControl corresponds with classical client server architecture. The services are stateful and multithreaded. The architecture and design of the whole application is strictly layered. All layers are well encapsulated with interfaces. Injection of the required “service”-object into the “client”-object is supported all over.

3.1 Decomposition

Next diagram shows the modular decomposition of the whole application. A short description of the components is written afterwards.

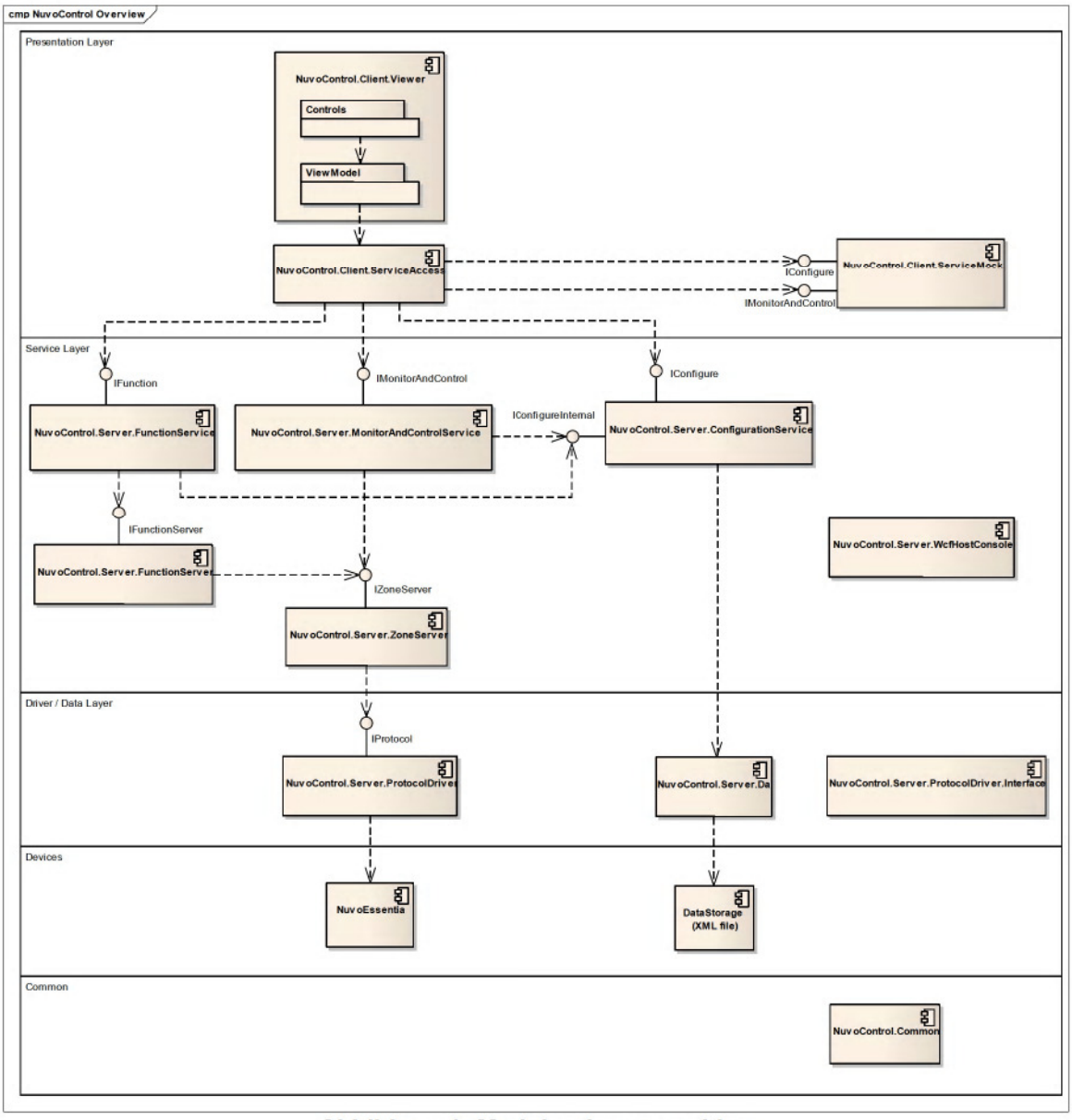


Abbildung 1: Modular decomposition

Der Begriff „application logic" kommt sehr oft vor. Für mich ist er ein Synonym zu „Business Logic" und sollte daher nur dafür gebraucht werden. In nachstehenden Beschreibungen wird der Begriff in einem anderen Sinne verwendet oder nicht?

3.1.1 Client-Side Components

Component:	Description:
NuvoControl Client.Viewer	This component contains the UI elements definition and the corresponding application logic. The component contains a couple of namespaces, whereas the shown above are the most important: • Controls: Contains the XAML definition of the UI elements and the code-behind files. Code-behind files may contain UI logic (handlers) and some application logic. However, as these classes are tightly coupled with the UI elements, the goal is, to have such functionality (application logic) one layer below. • View Model: Contains the application logic, like navigation between views. Subscribing for notification for zones shown in a view… It is important to see, that this layer has no dependency to the above (Controls). *See chapter 4 for more information.*
NuvoControl Client.ServiceAccess	This component manages the connection the services. It takes care, that session leases are renewed periodically. *See chapter 5 for more information.*

Tabelle 2: Client-Side Components

3.1.2 Server-Side Components

Component:	Description:
NuvoControl.Server.WcfHostConsole	This is the WCF host of all server objects. It is responsible for the lifetime of all services and the underlying servers (zone server, function server) as well as the protocol driver. *See chapter 14 for more information.*
NuvoControl.Server.ConfigurationService	The configuration service provides functionality related to the system configuration of NuvoControl; such as reading configuration or adding new functions. The configuration service is a WCF service, and is specified to be a singleton. *See chapter 6 for more information.*
NuvoControl.Server.MonitorAndControlService	The monitor and control service provides functionality to control and monitor NuvoEssentia zones. Thus, it contains methods to read zone states, command zone states or subscribe for zone state change notifications. To do so, it uses the zone server. The monitor and control service is a sessionful WCF service. Thus it keeps the context to its client (precisely: client proxy).

	See chapter 7 for more information.
NuvoControl.Server.FunctionService	To be defined. Not yet implemented.
NuvoControl.Server.ZoneServer	The zone server provides an image of the connected devices (NuvoEssentia) with their zone state. It is the client of the protocol driver. *See chapter 8 for more information.*
NuvoControl.Server.FunctionServer	To be defined. Not yet implemented.
NuvoControl.Server.ProtocolDriver	Implements the protocol stack to provide communication facility with the devices (NuvoEssentia). *See chapter 11 for more information.*
NuvoControl.Server.ProtocolDriver.Interface	Defines the interface of the protocol driver. Contains the factory method for the dynamic creation of the driver, dependent on the connected device types. *See chapter 12 for more information.*
NuvoControl.Server.Dal	Encapsulates the persistent system configuration. The store of the system configuration is an XML file. Converts the XML data into system configuration objects (data transfer objects). These objects are used on client and service side. *See chapter 10 for more information.*

Tabelle 3: ServerSide Components

3.1.3 Common Components

There is one common component which is deployed on client as well as on server side:

Component:	Description:
NuvoControl.Common	Defines common classes and functionality used on client and server side. In general these are the system configuration classes (data transfer objects) and the zone state. *See chapter 6 for more information.*

Tabelle 4: Common Components

3.2 Client-Server Interfaces

Next table shows the interfaces which are exposed to clients. Detailed description of the functionality can be retrieved in code or out of the generated help file.

Interface:	Description:
IConfigure	Defines functionality to read and modify the system configuration of NuvoControl.
IMonitorAndControl	Defines functionality to control and monitor zones of connected devices (NuvoEssentia).
IFunction	To be defined. Not yet implemented.

Tabelle 5: Client-Server Interfaces

3.3 Client-Server Communication

A client has following requirements on the service regarding the communication patterns:

- Calls with short duration shall be synchronous (RPC style, synchronous request-reply). This allows for easy implementation on the client side. Nearly all methods of the service interfaces can match this requirement.
- Calls with long duration shall be asynchronous (fire and forget). There are two possibilities to support this scenario:
  - The client calls the method synchronously on a separate worker thread. (Note that this pattern is greatly supported by the .NET framework).
  - The service supports callbacks on separate threads.
- A subscription pattern must be implemented. Thus the client can subscribe itself for certain notifications of the service. Thus, the service must support callbacks in separate threads.

Further optional requirements on the interface:

- The interface shall be platform neutral. Thus a client, written on any platform can use the interface.

WCF (Windows Communication Framework) is a communication framework, supporting various so-called bindings. A binding is merely a consistent, canned set of choices regarding the transport protocol, message encoding, communication pattern, reliability, security, transaction propagation, and interoperability.
Next table summarizes the important bindings with their capabilities:

Note that the table does not consider:

- Federation
- Peer networking
- And some other esoteric mechanisms like JSON… ☺

Binding	Transport	Encoding	Inter-operable	Callbacks	Transport Level Session	Message Reliability	Ordered Delivery	Queuing	Transaction	Security
BasicHttpBinding	HTTP/HTTPS	Text/MTOM	Yes	No	No	No	No	No	No	No
WSHttpBinding	HTTP/HTTPS	Text/MTOM	Yes	No	Yes (if security or reliability is enabled)	Yes	Yes	No	Yes	Yes
WSDualHttpBinding	HTTP	Text/MTOM	No	Yes (duplex channel)	Yes (if security or reliability is enabled)	Yes	Yes	No	Yes	Yes
NetTcpBinding	TCP	Binary	No	Yes	Yes	Yes	Yes	No	Yes	Yes
NetMsmqBinding	MSMQ	Binary	No	Yes	Yes	Yes	Yes	Yes	?	?
NetNamedPipeBinding	IPC	Binary	No (only same machine)	Yes	Yes	Yes	Yes	n.a.	n.a.	n.a.

Tabelle 6: Bindings and its capabilities

According to above table and the requirements written before, it is quite easy to choose the appropriate binding.
Possible bindings are:

- WSDualHttpBinding
- NetTcpBinding
- NetMsmqBinding

With all of these bindings, we can not fulfill the optional requirement of interoperability.

schön, diese Vergleichstabelle

- We choose **WSDualHttpBinding** for services which need to support callbacks, as this is the most open standard of these three.
- We choose **WSHttpBinding** for the configuration service.

3.4 Hosting

Service hosting is possible with three different approaches:

- Self-Hosting
- IIS Hosting
- WAS (Windows Activation Service)

We choose **self-hosting**, because of its simplicity.

3.5 Overview of the Service Layer

Next diagram shows the classes of the service layer and its associations. The MonitorAndControlService implements the interface IMonitorAndControl which is exposed to the client. It uses the ZoneServer for reading and commanding zone states. The ZoneServer keeps all ZoneControllers, which use the protocol driver to communicate with their appropriate 'real' zone.

The ConfigurationService implements the IConfigure interface, which is exposed to the client. It uses the ConfigurationLoader to read and write the configuration from and to XML file.

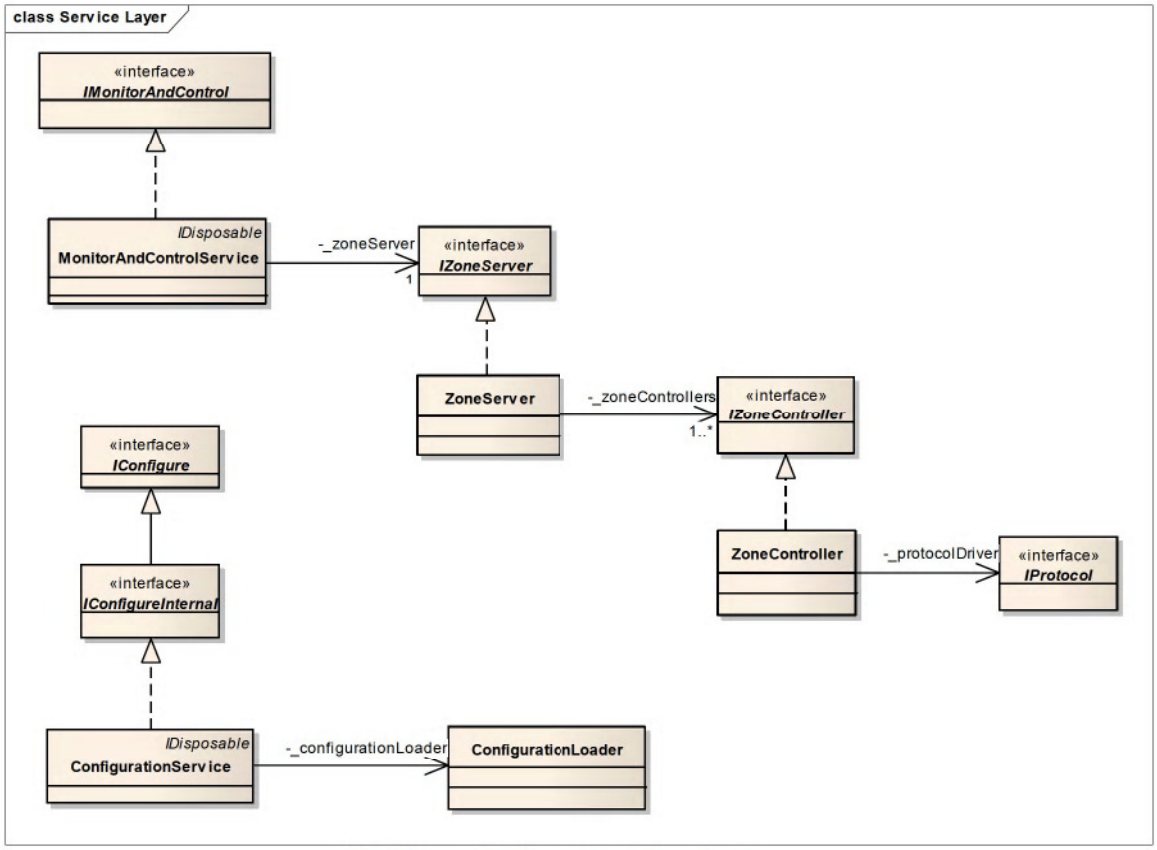


Abbildung 7: Overview Service Layer

Next diagram shows an object diagram of the service layer for a scenario with three connected clients and four available zones:

- As the M&C service is sessionful, there exist three objects of class MonitorAndControlService.
- The ZoneServer is instantiated exactly one time.
- For each zone, a ZoneController object is created.
- The ConfigurationService is configured as singleton, thus only one object exists.

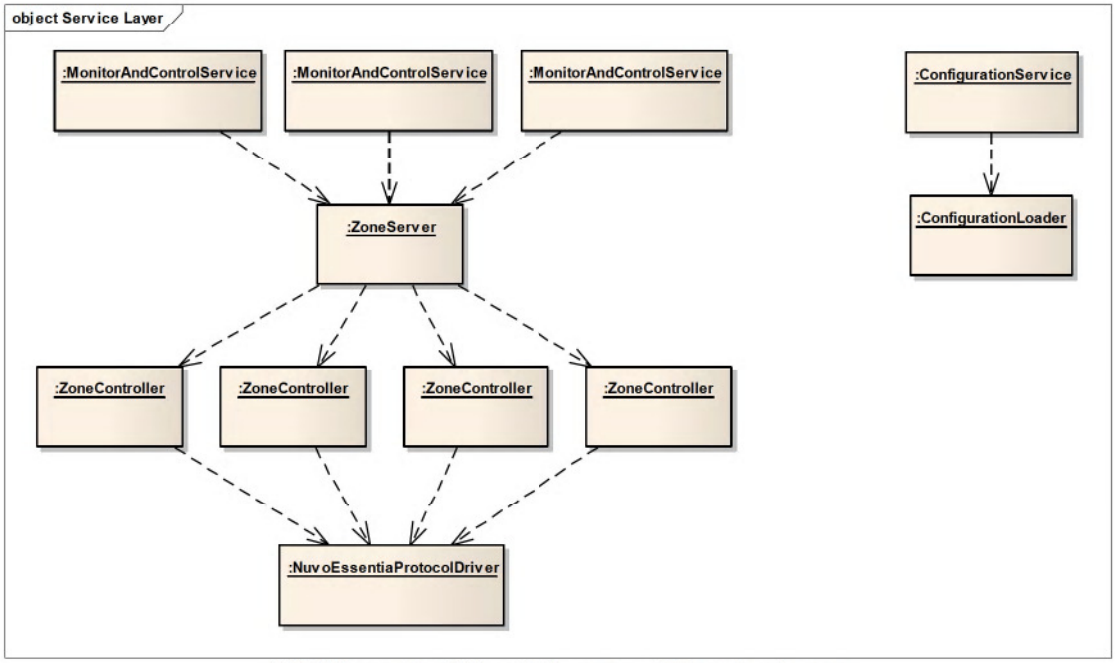


Abbildung 8: Object Diagram of Service Layer

3.6 Startup and Shutdown

Next diagram shows the startup sequence of the whole server. Any description is added with notes into the diagram.

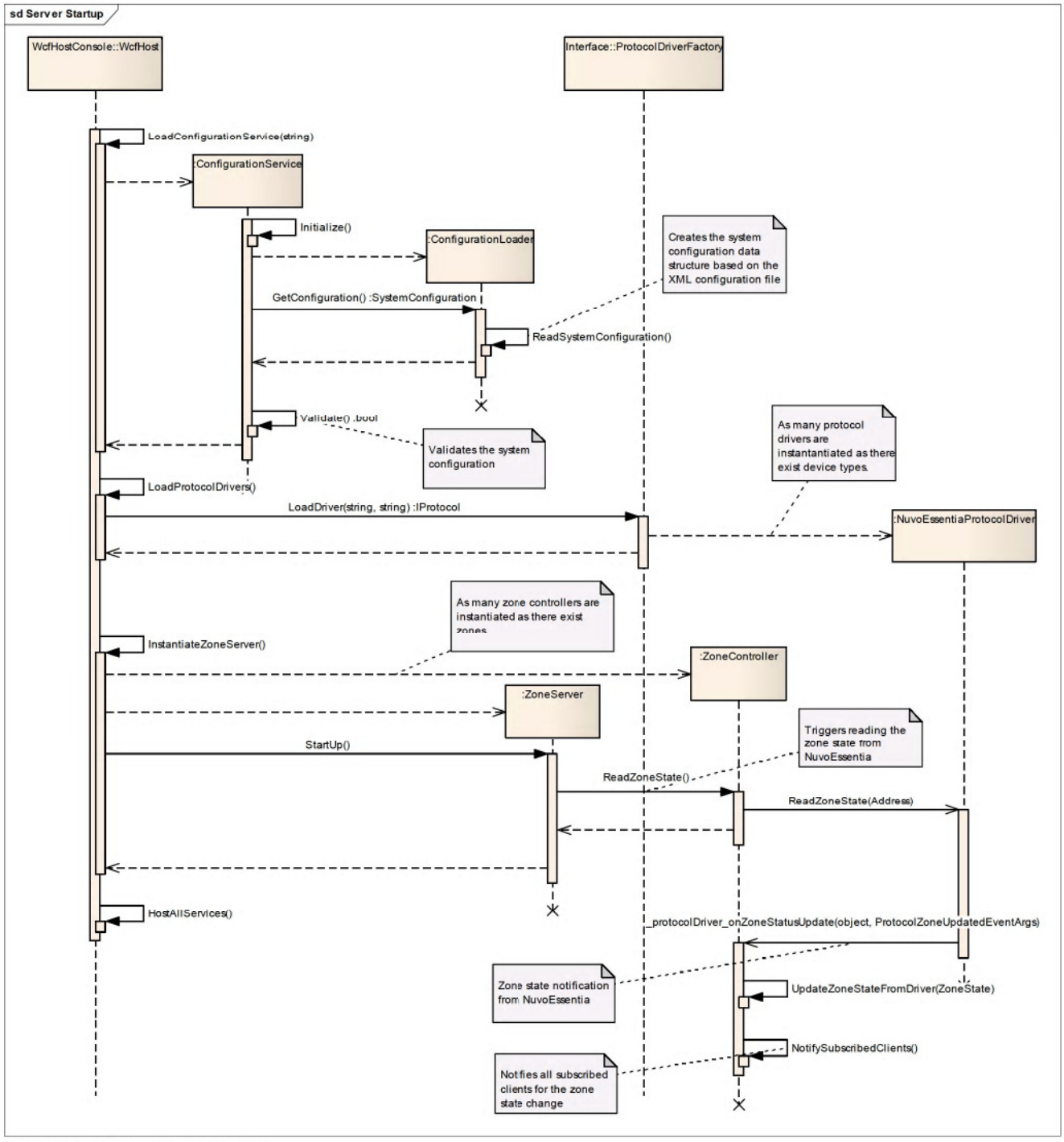


Diagram 9: Server Startup

(sehr schön, auch asynchrone Messages berücksichtigt)

Till now, the shut down of the server can be kept surprisingly easy. So far, we do not modify the configuration (e.g. functions) at runtime and therefore do not need to persist such changes. The server does also not have a logical connection or subscriptions to other external components/services. Thus we simply can close the WCF service hosts and leave the Main() of the console host.

3.7 Multithreading

WCF supports three concurrency modes for the hosted objects. These are:

- Single; all interface methods are synchronized. Callbacks within service calls on the same thread are not allowed.
- Multiple; the interface methods are not synchronized.
- Reentrant; the interface methods are synchronized. It is allowed to do callbacks within a service call on the same thread (however, doing so, any state of the service must be consistent in this moment!!!).

We choose the single concurrency mode. Thus WCF synchronizes calls on the service hosted objects. These are the ConfiguationService and the MonitorAndControlService objects. (Note, that the objects are synchronized!)
Thus, as the zone server is used by various objects (MonitorAndControl, later also the FunctionServer) we must still:

- Synchronize the access to the ZoneServer,
- Synchronize the access to the ZoneControllers, they are used by ZoneServer and Protocol driver.
- Synchronize access to further state of the protocol driver, which may be a access by more than one thread.
- Introduce patterns to prevent deadlocks.

gut

hm, das sieht man in Abb. 8 nicht

To prevent deadlocks we take the approach to access the objects (to be synchronized) in a well defined order. The order shall be the same order as an incoming call to read the state of a zone (GetZoneState()) is these objects:
1. MonitorAndControlService;
2. ZoneServer;
3. ZoneController
4. Protocol Driver

Thus the implication on this is, that the notifications coming up from the protocol driver to the zone controller and further up to the MonitorAndControlService are not allowed to be fired in a lock of the according object.

3.8 Error Handling

To be done

3.9 Unit Testing

The architecture and design of the whole application is strictly layered. All layers are well encapsulated with interfaces. Injection of the required “service”-object into the “client”-object is supported all over. E.g. The MonitorAndControlService class supports injection of the underlying Zone

Server; the ZoneServer supports injection of the underlying ZoneControllers; This is true as well for the whole protocol stack.
Thus this design greatly supports unit testing of all layers and components. Worth mentioning is especially the extensive unit tests for the protocol driver stack.
Also on client side, most of the application logic (which resides in the View Model layer) is unit testable by means of injection of a ServiceMock object (simulating the service).

4. NuvoControl.Client.Viewer

4.1 Overview

The NuvoControl viewer is a full-blown WPF application, allowing navigating through the building, the floors and of course the zones. Zones can be commanded and their state is visualized. Next screenshots give a better impression of the functionality.

The application window is in code defined by the *MainWindow.xaml*. It is divided into four areas:

- A conventional menu bar, containing all possible commands. Note, most of the command do support shortcuts.
- The navigation bar; it allows the navigation between the different view types. Within the same view, it allows navigating between different view representations.
- The application view
- The status bar.

The main view is the top view of the application. In code, this view is defined by the *MainView.xaml*. Next picture shows the main view.

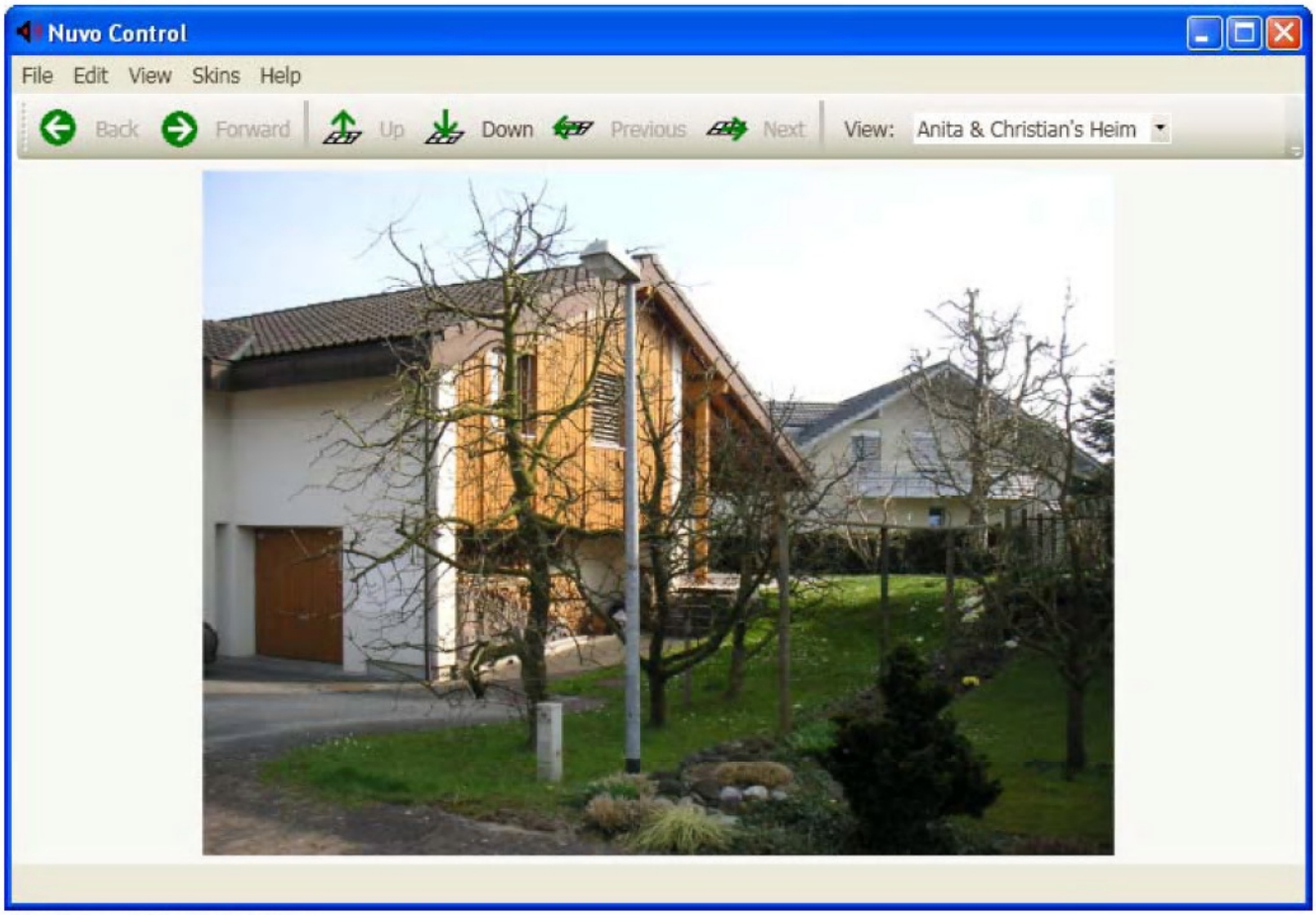


Abbildung 10: Main View

Clicking the “Down” button on the navigation bar brings up the floor view. In code, this view is defined by the *FloorView.xaml*.
The floor view shows all zones of a floor. For each zone, the state is visualized. With a popup, the user may command an individual zone. In code, a zone is defined by the *ZoneControl.xaml*. The state info is defined by the *ZoneInfo.xaml*, the content of the popup is defined by the *ZoneCommander.xaml.*

With the “Next” and the “Previous” button, the user may navigate other floors (a different view representation). With the “Up” button, navigation to the MainView occurs. With the “Down” button, navigation to a single zone occurs. The drop down lists box allows immediate selection of a floor.

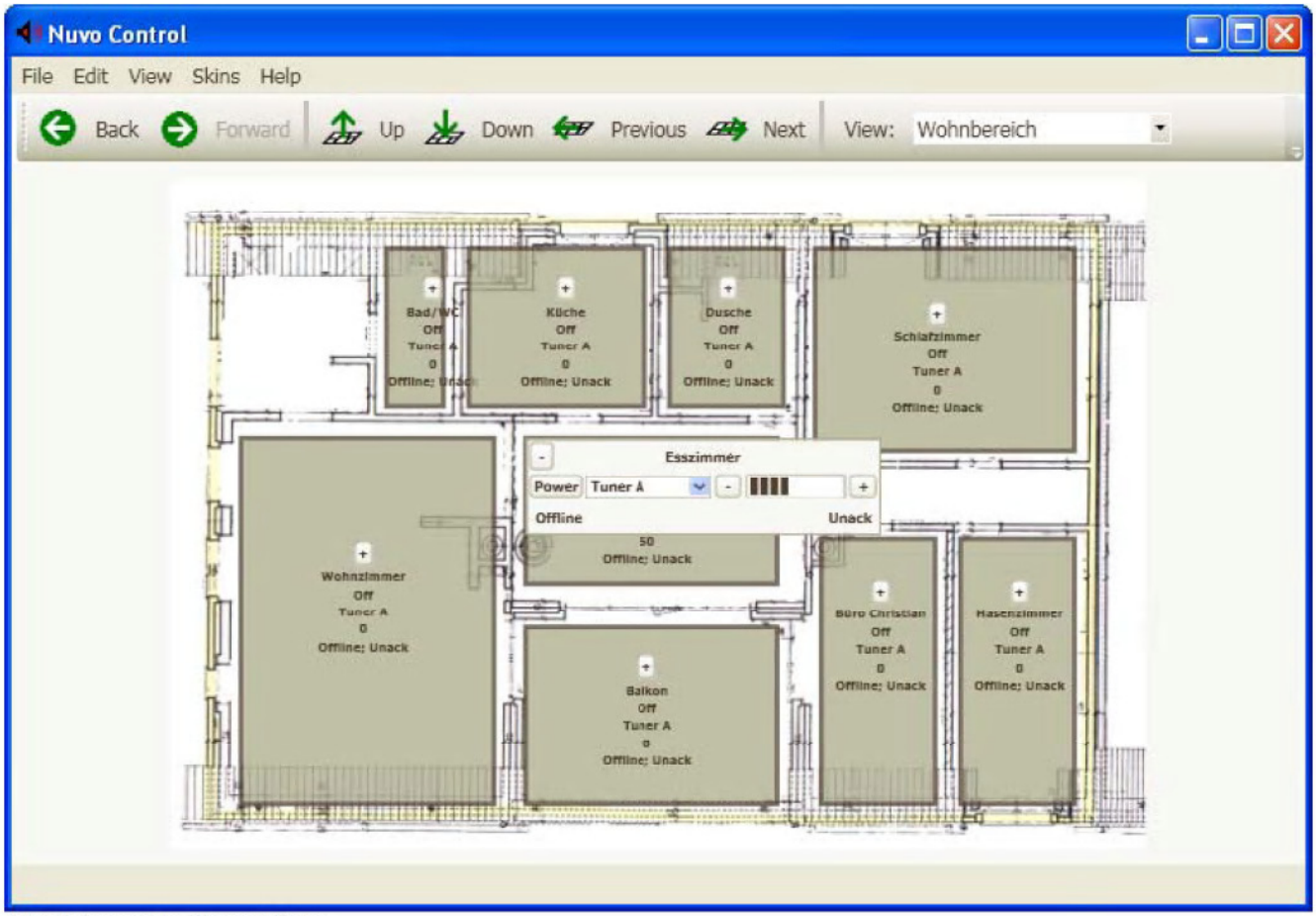


Abbildung 11: Floor View

Next picture shows the zone view. In code, this view is defined by the *ZoneView.xaml*. Navigation is analogous to the description above.

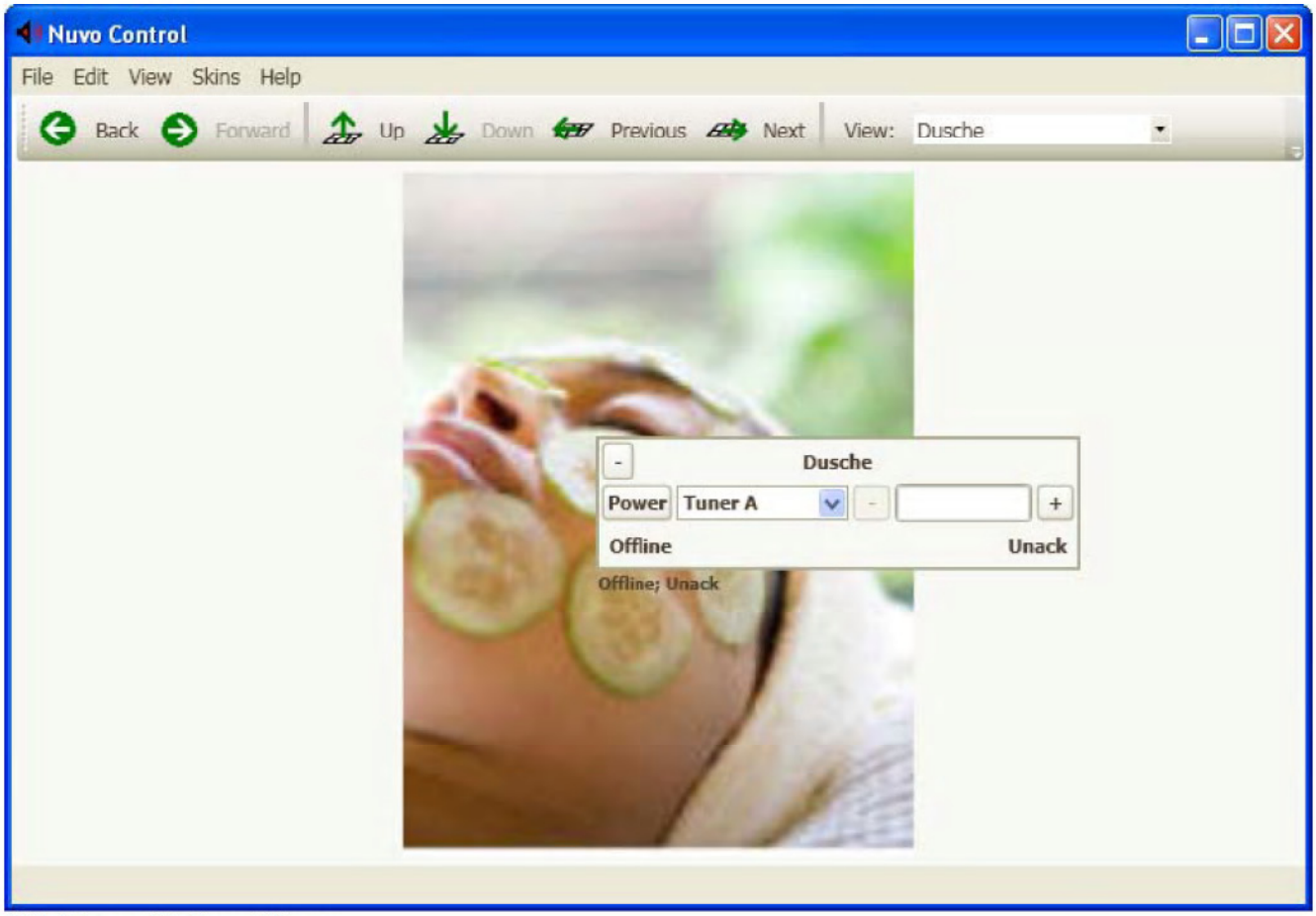


Abbildung 12: Zone View

4.2 Used Concepts

The application uses a fair amount of new (and of course old) UI concepts introduced by WPF. Such as:

- Definition of the UI (UI elements and their layout) in XAML.
- Data Binding.
- User Controls.
- Dependency Properties.
- Commands.
- Styles, Triggers and Animation
- Shapes and Brushes.

These concepts should be known by the reader of next chapters. However, following table shall give a very brief idea what they are.

Concept:	Description:
XAML	XAML is a XML language, used to define the composition and definition of UI elements in an application. Furthermore, it allows to define bindings, styles, animations, and more…
Data Binding	In WPF, properties of a UI element (e.g. the text of a label) can be bound to a property of an underlying C# data object. Thus, the property of the UI element (e.g. the text the label) represents the property of the data object. Further mechanisms support notifications of the UI element upon changes of the data object. Advantage: No dependency of the data object to the UI element.
User Controls	Collection of UI elements. Reusable component. This concept is supported since WinForms and ASP.NET
Dependency Properties	Dependency properties support change notifications and property value inheritance. Dependency properties are the basis for supporting concepts like data binding, styles and animation.
Commands	Commands and CommandBinding are used, to support having command handler at one place (ViewModel) and being able to invoke the command anywhere in UI layer.
Styles, Triggers and Animation	Styles allow designing the visual appearance (colors, shades, margins, paddings, animations …) completely independent from the code. It is a similar concept as the CSS in HTML. Triggers define when a style shall change. E.g. may define a style change upon change of a property of a data object. Animations support continuous change of dependency properties. This allows defining effects like changing the background color or the opacity over a time frame.
Shapes and brushes	Shapes and brushes are visual elements.

Tabelle 13: Used UI Concepts

4.3 Design

The viewer is divided into following namespaces:

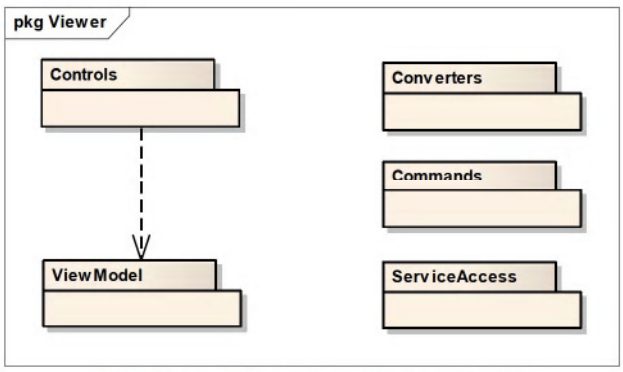


Abbildung 14: Viewer Namespaces

Namespace:	Description:
…Controls	Contains the XAML files for the UI definition and the code behind files. The overview of this chapter, describes where in the GUI which controls appears.
…ViewModel	Contains most of the UI and application logic.
…Converters	Contains value converters uses for data binding.
…Commands	Defines the application commands (navigation commands and zone commands).
…ServiceAccess	Contains a configuration object, used to start up the viewer in test mode or in real runtime mode.

Tabelle 15: Viewer Namespaces

The next class diagram shows the view model.

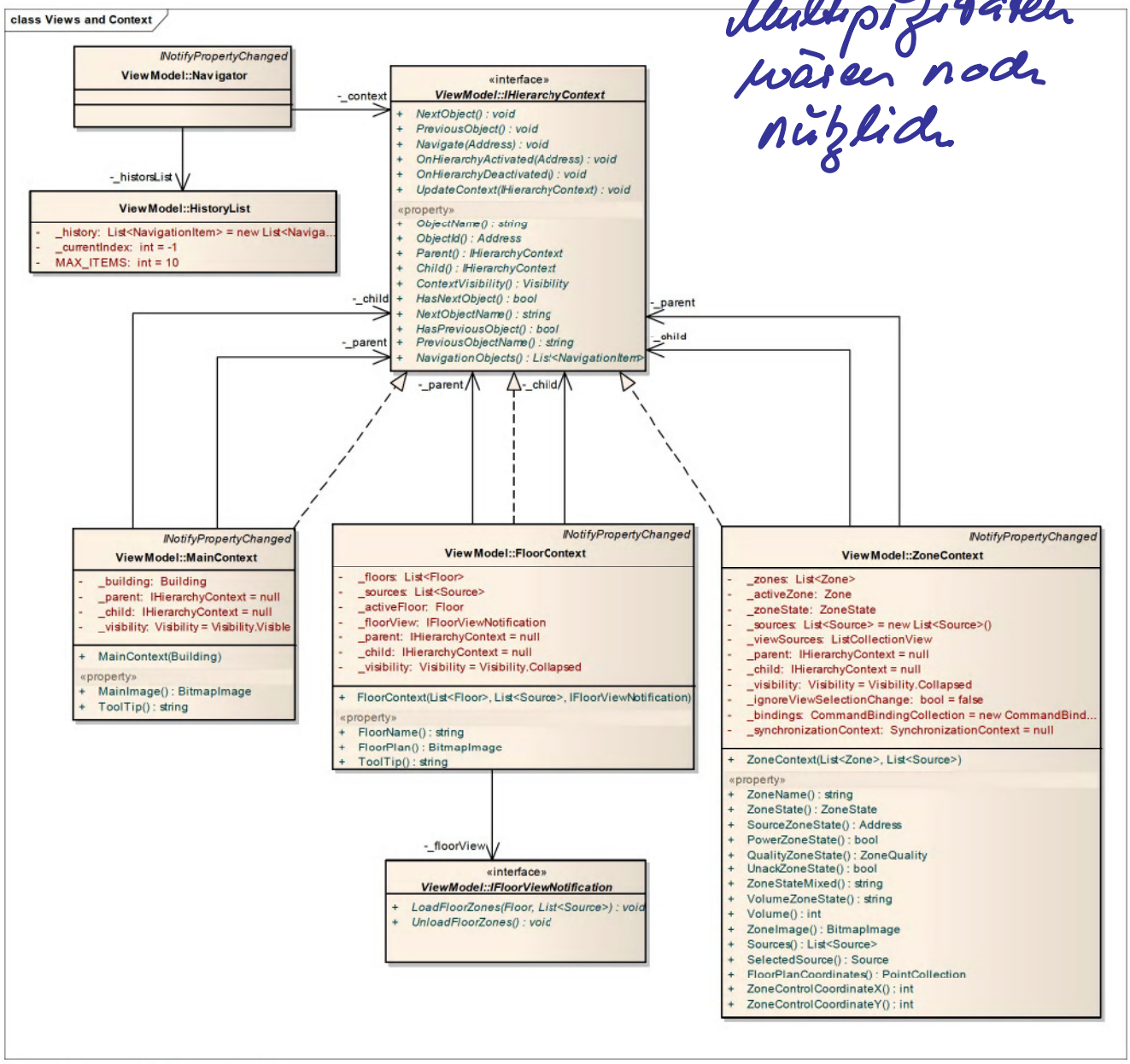


Abbildung 16: View Model

Class:	Description:
Navigator	The navigator manages the navigation between the view types (Main view <-> floor view <-> zone view. Thus is contains the command handler for the navigation commands. The navigator also maintains the history list of the visited views. The navigator is the data context for the MainWindow.xaml. Thus, the menu and the navigation bar retrieve its tooltips via data binding.
HistoryList	Stores the "visited views". Provides the information for the navigator to browse forward and back in the history.
IHierarchyContext site	An interface, which the data context of a view type needs to implement. The current design of the view model would allow introducing more view types. E.g a view showing the areal of a company, consisting of several buildings. Thus, any data context associated with a view type, implementing this interface can participate in the navigation mechanism of the application. The only addition requirement is, that the configuration file describes the association between the areal and probably the buildings in it.
MainContext	This is the data context of the main view (MainView.xaml). It provides the main image, the name of the view, a tooltip... via databinding to MainView.xaml. The child of this context is the floor context.
FloorContext	This is the data context of the floor view (FloorView.xaml). It provides the floor plan image, the name of the view, a tooltip... via databinding to FloorView.xaml. Note, here there is a exceptional behaviour: The floor view contains various zone controls. It would be nice, if these zone controls also would be instantiated via data binding. The concept of data templates would most probably serve here. However, due to lack of time, we tool another approach: The FloorView.xaml.cs implements the interface IFloorViewNotification. This implementation unloads and loads zones within a floor view. With this approach, we still do not have a dependency to the controls layer.
ZoneContext	This is the data context of the floor view (FloorView.xaml). It provides the zone image, the name of the zone, a tooltip... via databinding to ZoneView.xaml. It also contains the application logic to set, read and monitor zone state of the service.

Tabelle 17: View Classes

Note, that the context classes together model the composite pattern.

Object diagram:

An object diagram would show that there exists exactly one object for the Navigator, the HistoryList, the MainContext and the FloorContext. Thus, if the floor is changed in the view, the floor context changes its _activeFloor field, which points to desired floor configuration object.
The same is **not** true for the ZoneContext. If the current view is showing a floor, there are as many ZoneContexts as zones exists in this floor. These zone contexts are bound to the corresponding zone control (ZoneControl.xaml). However, if the view is showing a zone, there is only one ZoneContext object which is bound to the ZoneView.xaml.

Bis hier im Detail gelesen, Rest überflogen.

For further understanding of the design, the next sequence diagram shows the scenario when the user navigates from top view to a floor view and further down to the zone view.

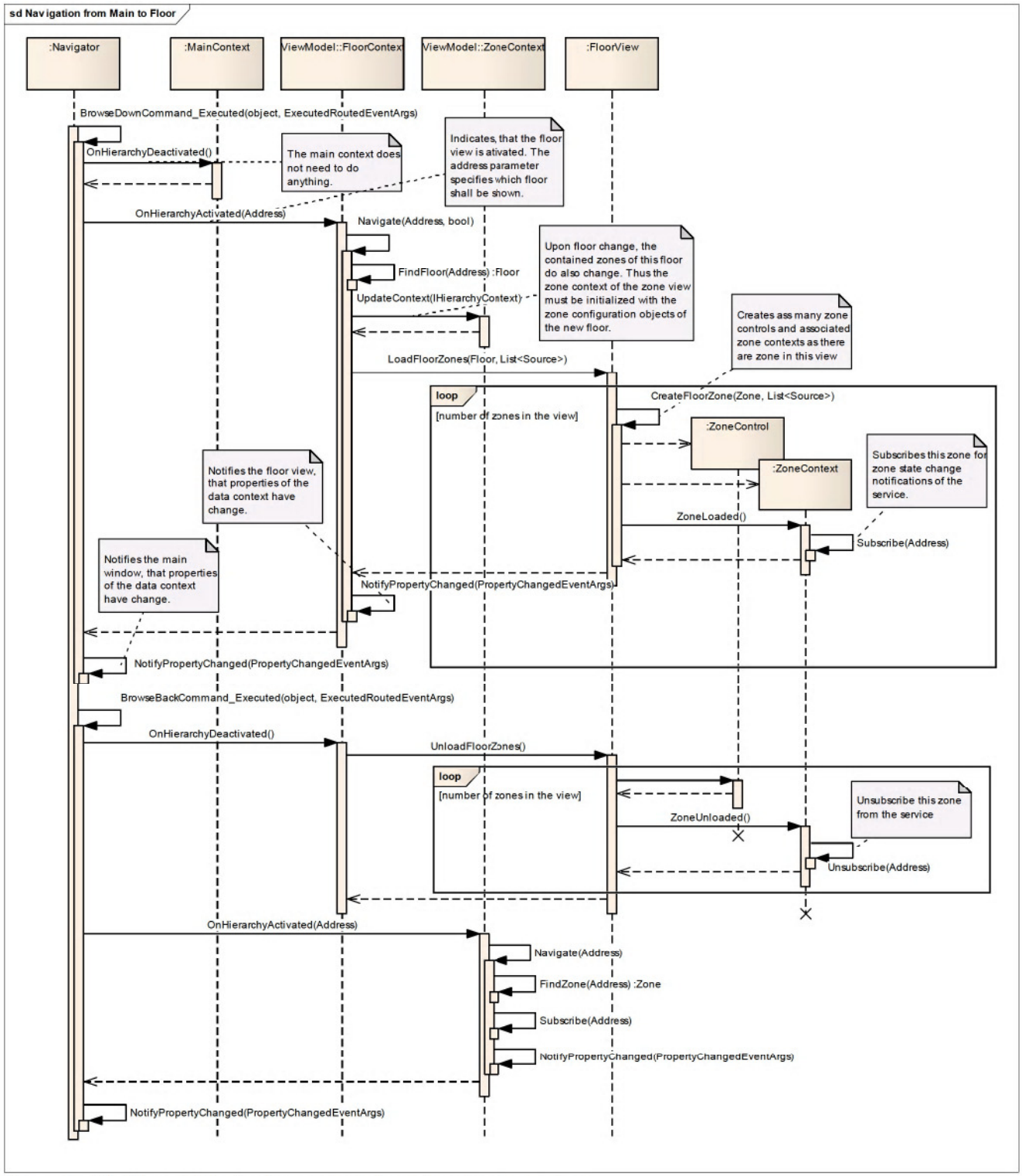


Diagram 18: Viewer Navigation – from top view to zone view

4.3.1 Subscriptions

It's worth mentioning, that subscribing and unsubscribing for zone state change notifications is done according to the zones visible in the actual view. In other words, only the zones are subscribed which are shown in the current view.

4.3.2 Startup and Shutdown

Upon the window loaded event, the service proxies are instantiated and the configuration data is read from the service. According to the configuration data, the views and their data contexts are instantiated.

Upon the window unloaded event, the service proxies are disposed. Note that the service handles remaining subscriptions of the client properly by himself.

4.3.3 Skins

Five different visual appearances are designed with style. These skins can be dynamically changed via menu. Next picture show one of them.

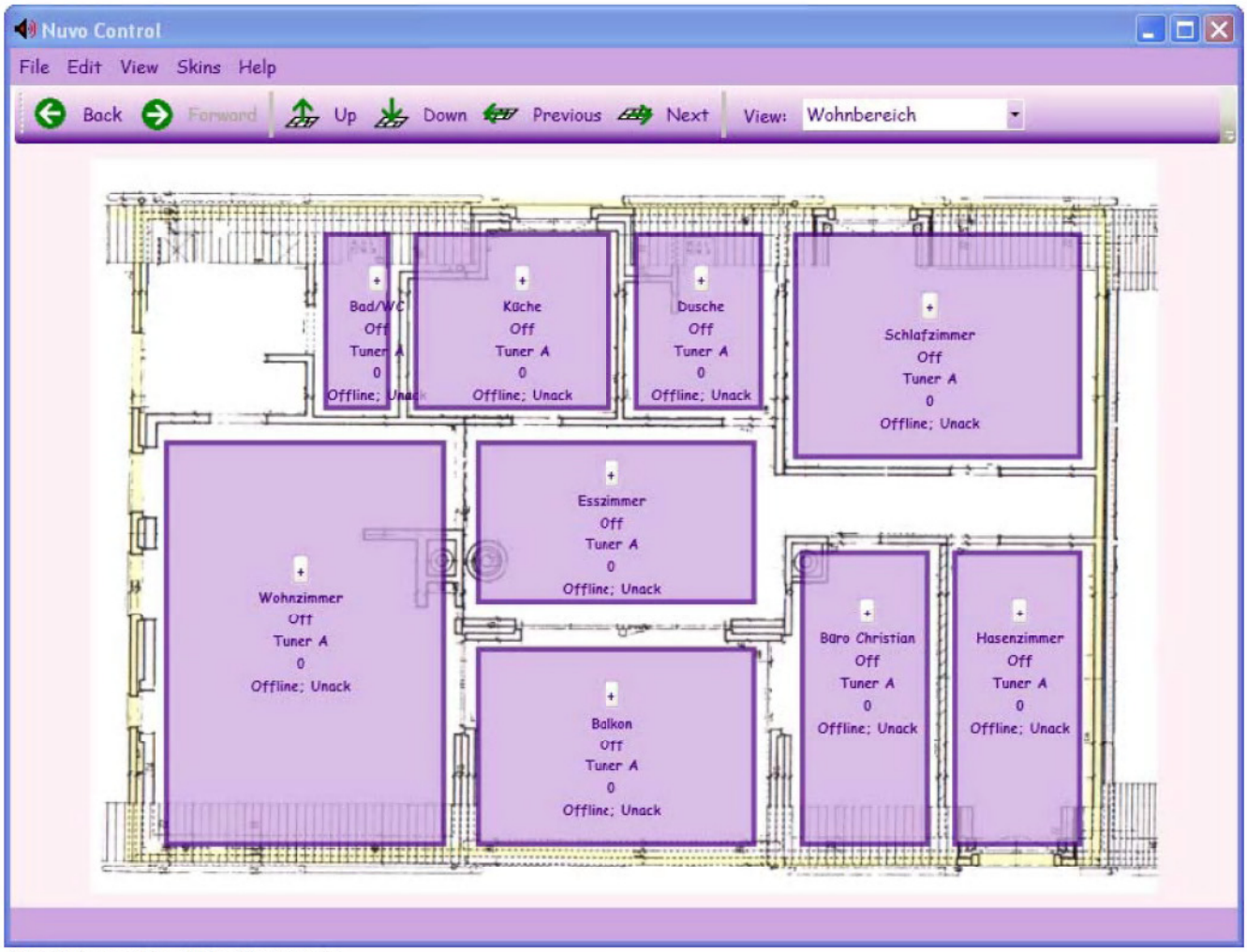


Abbildung 19: Skin Example

5. NuvoControl.Client.ServiceAccess

This component manages the connection the services. It takes care, that session leases are renewed periodically.

Next diagram shows the classes of this component.

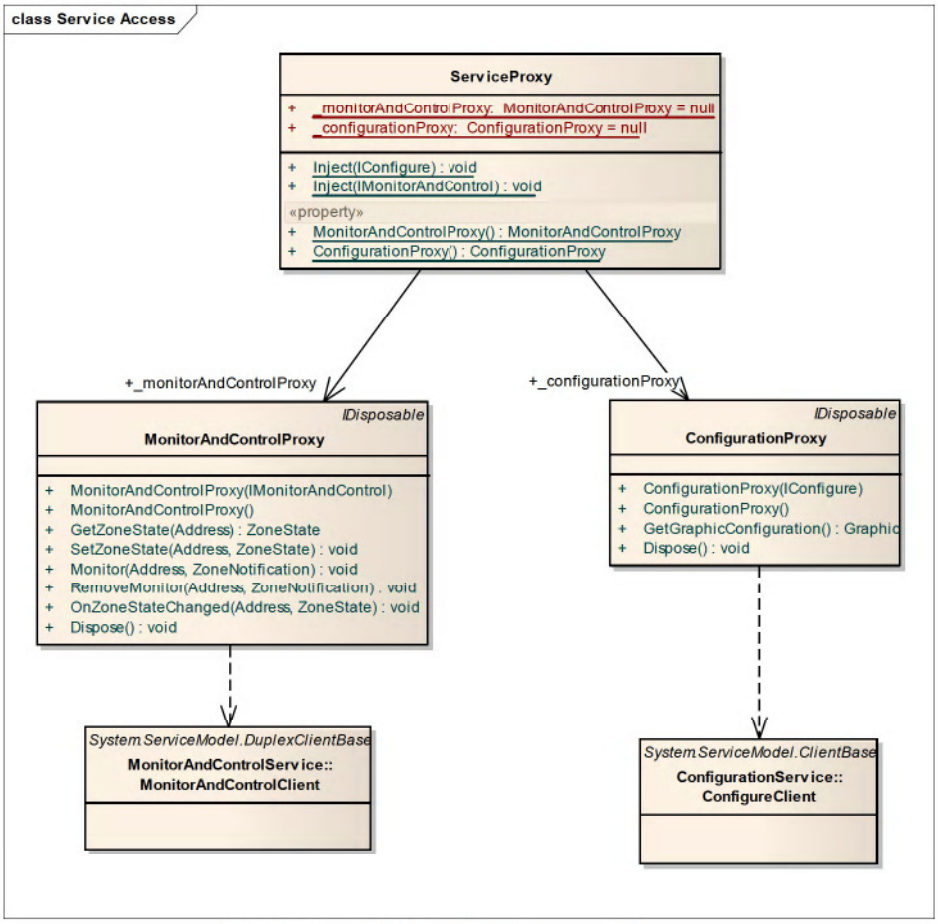


Abbildung 20: Service Access Classes

Class:	Description:
MonitorAndControlClient	The WCF generated service proxy to access the MonitorAndControlService.
ConfigurationClient	The WCF generated service proxy to access the ConfigurationService.
MonitorAndControlProxy	A wrapper around the WCF proxy. Supports injection of a mock proxy for tests mode. Manages the renewal of the service session lease.
ConfigurationProxy	A wrapper around the WCF proxy. Supports injection of a mock proxy for tests mode. Manages the renewal of the service session lease.
ServiceProxy	A very thin wrapper for accessing service functionality via static methods within the zone contexts and and the window loaded and unloaded events.

Tabelle 21: Service Access Classes

6. NuvoControl.Common (Configuration)

This component defines common classes and functionality used on client and on server side. In general these are the system configuration classes and the zone state. For a description of these classes see the generated help file.

Next diagram shows all classes of the system configuration of NuvoControl with their dependencies to each other.
The system configuration describes:

- Which hardware (devices) is connected to the application and what zones and sources exists per device. For each device, the protocol driver and the communication parameters are defined.
- Zero or more functions (alarm functions or sleep functions). A sleep function is associated with a zone. An alarm function is associated with a zone and a source.
- The graphical configuration. It describes the graphical attributes of the building with its floors and the zones of the floors. It also describes the graphical attributes of the available sources

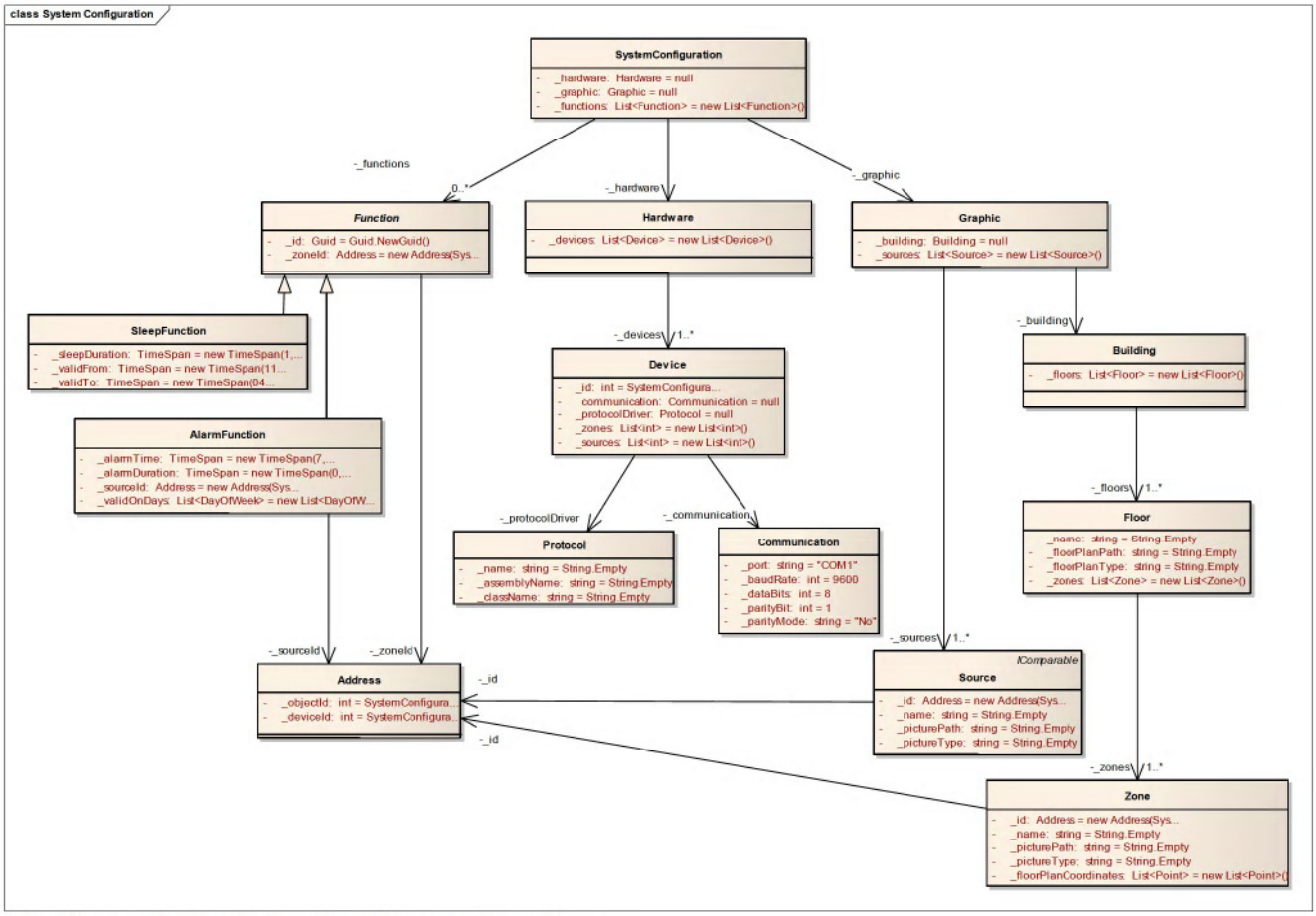


Abbildung 22: Configuration Classes Overview

7. NuvoControl.Server.MonitorAndControlService

The monitor and control service provides functionality to control and monitor NuvoEssentia zones. Thus it contains methods to read zone states, command zone states or subscribe for zone state change notifications. To do so, it uses the zone server.
The monitor and control service is a sessionful WCF service. Thus it keeps the context to its client (precisely: client proxy).
For a description of the classes see the generated help file.

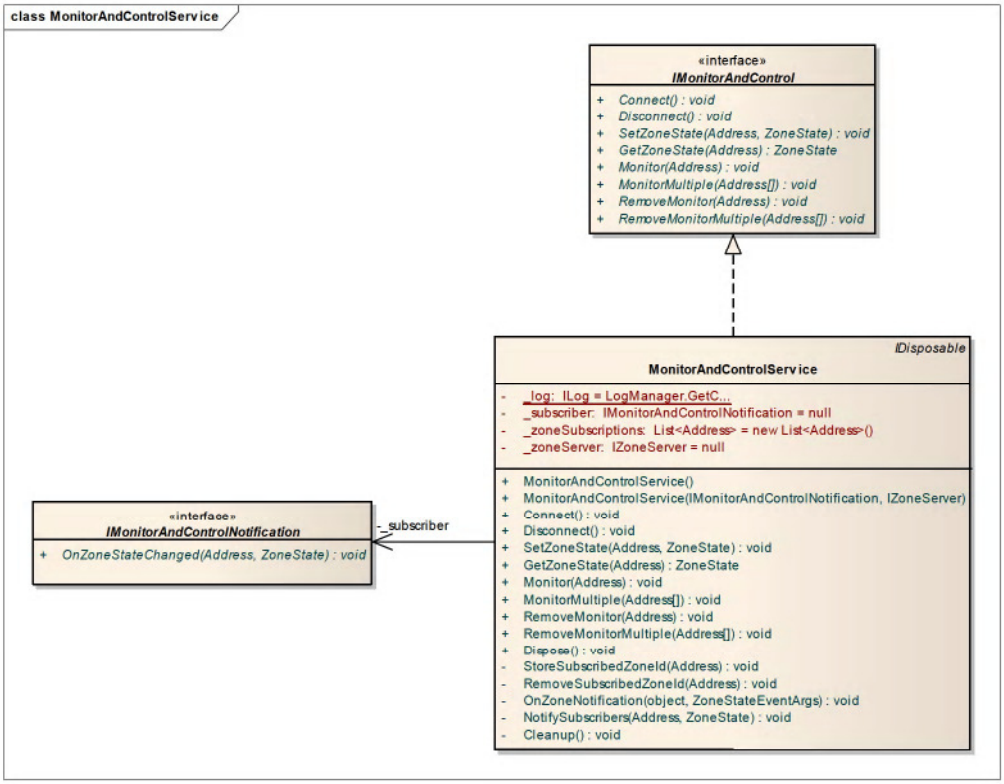


Abbildung 23: Monitor and Control Classes

8. NuvoControl.Server.ZoneServer

The zone server provides an image of the connected devices (NuvoEssentia) with their zone state. It is the client of the protocol driver. It provides its interface to the monitor and control service and the function server. The zone server is instantiated once within the one server process.
For a description of the classes see the generated help file.

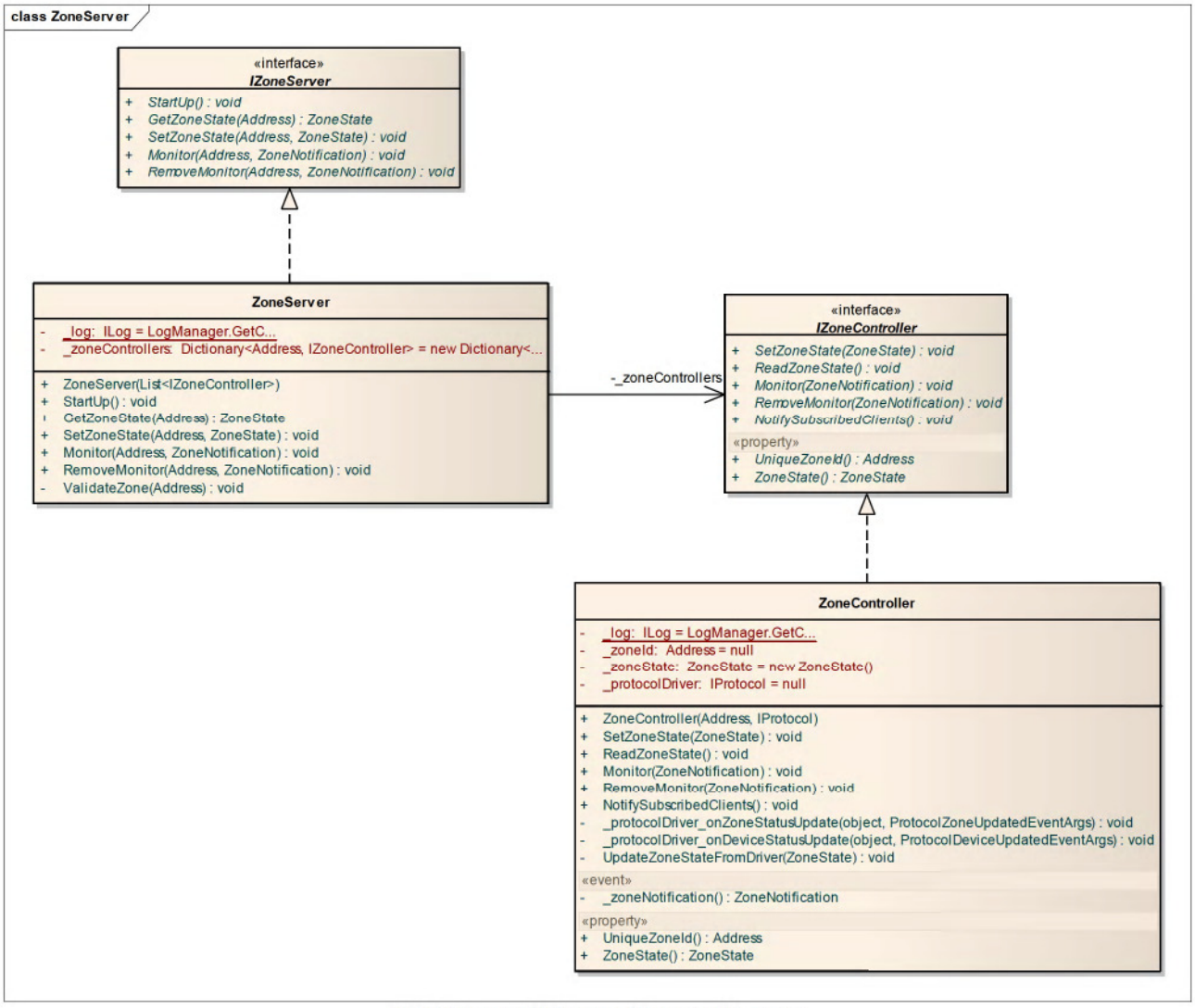


Abbildung 24: Zone Server Classes

9. Configuration Service

The configuration service provides functionality related to the system configuration of NuvoControl; such as reading configuration or adding new functions.
The configuration service is a WCF service, and is specified to be a singleton.
For a description of the classes see the generated help file.

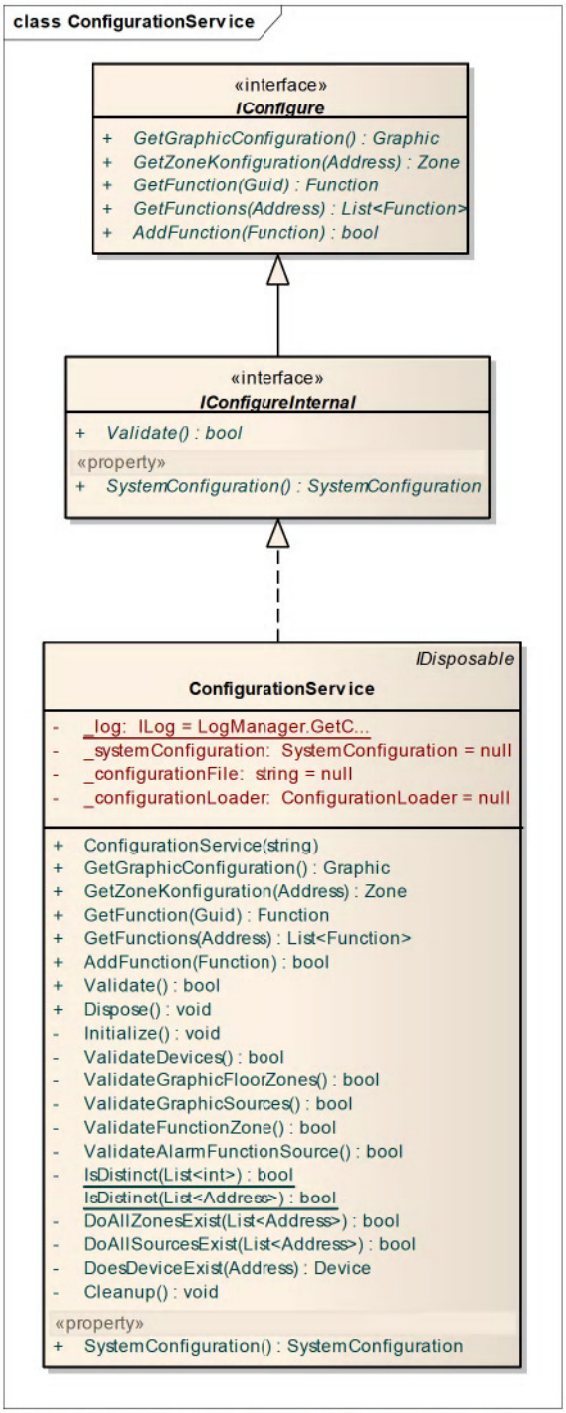


Abbildung 25: Configuration Service Classes

10. NuvoControl.Server.Dal

Encapsulates the persistent system configuration. The store of the system configuration is a XML file. Converts the XML data into system configuration objects (data transfer objects).

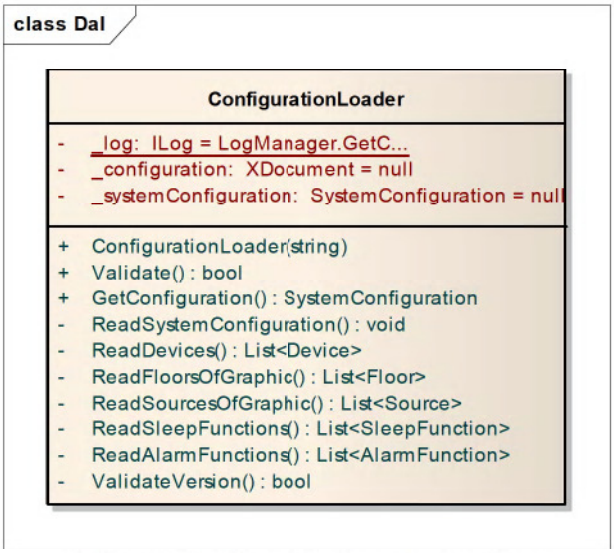


Abbildung 26: Data Access Layer Classes

11. NuvoControl.Server.ProtocolDriver

This namespace contains all concrete classes of the protocol driver. These classes' implements the protocol driver interface described in the next chapter.
For a description of the classes see the generated help file.

11.1 Protocol Driver Hierarchy

The following picture shows the most important classes of the protocol driver.

Abbildung 27: Protocol Driver Classes Hierarchy

11.2 Protocol Driver Commands

The class diagram on the next page shows some of the classes in more detail. The class NuvoEssentiaSingleCommand contains all functions to parse and create a command. It retrieves the parameters from the input string and makes them available via member functions. On the other side it creates also the outgoing string.

The class NuvoEssentiaProtocolDriver implements the non-device specific interface IProtocol. It also implements the device-specific extensions in INuvoProtocol.
The commands itself are defined in a separate xml file called NuvoEssentiaProfile.xml.

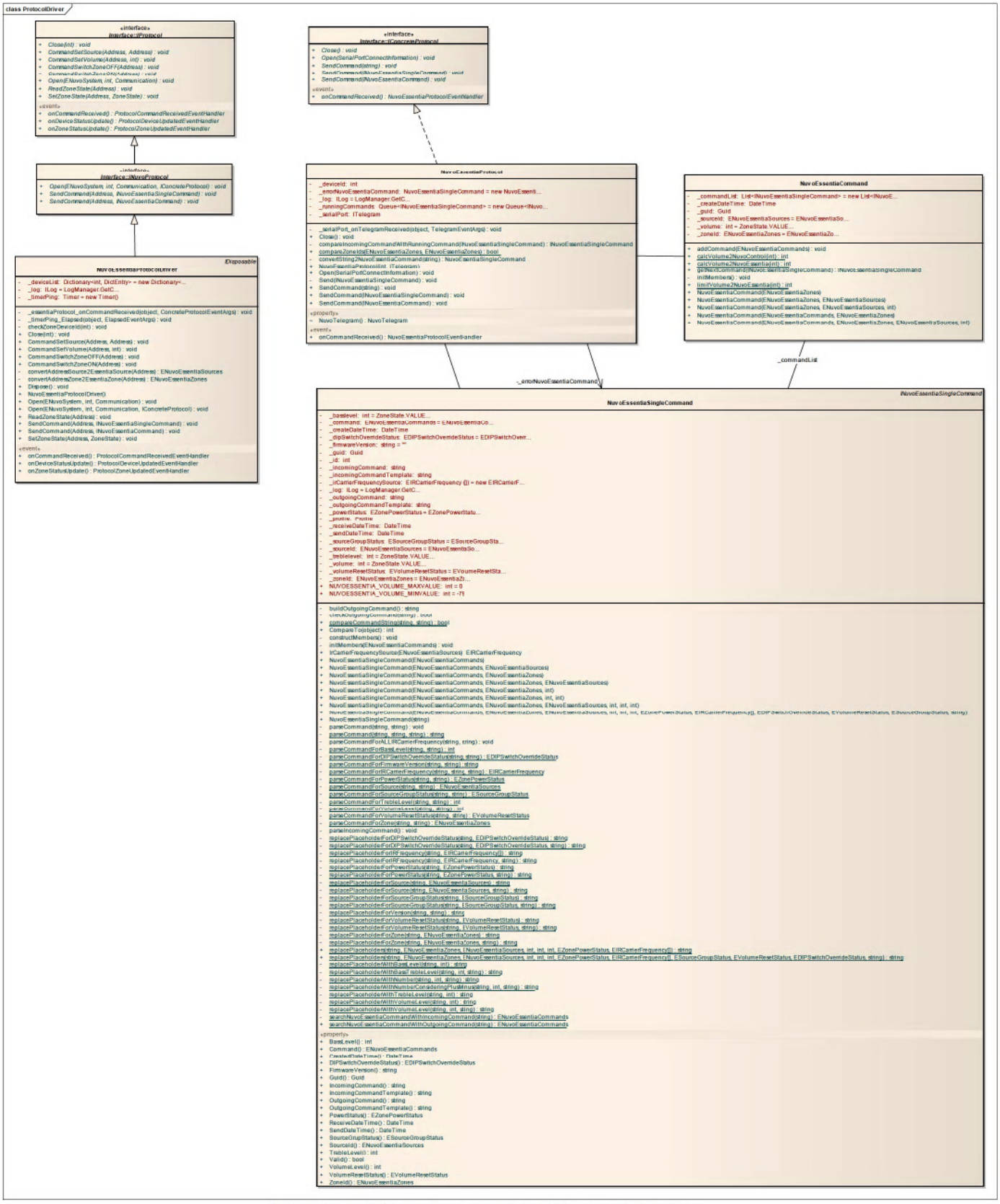


Abbildung 28: Protocol Driver Classes Details

11.3 Protocol Driver Serial Ports

The interface ISerialPort is not only used by the concrete serial port driver, it is used by several other classes for different purpose.

- SerialPort: This class implements the interface ISerialPort. It's the concrete serial port driver.
- SerialPortQueue: This class implements the interface ISerialPort. This class connects to a queue using MSMQ, instead to a real serial port. This mechanism is used to communicate with the standalone simulator.
- ProtocolDriverSimulator: This class implements the interface ISerialPort, to implement the in-built simulator. See chapter 13 for more details.
- SerialPortMock: This class implements the interface ISerialPort. It is used in the unit test as mock object.
- SerialPortFactory: This class uses the interface ISerialPort. (not fully implemented yet)

12. NuvoControl.Server.ProtocolDriver.Interfaces

This component contains all interfaces and factory classes used in the protocol driver. As described in the previous chapter, the protocol driver is strictly layered. Two layers are connected via an interface. For a description of the classes see the generated help file.

The following picture shows all interfaces together with its event argument classes and delegates defined in the protocol driver.

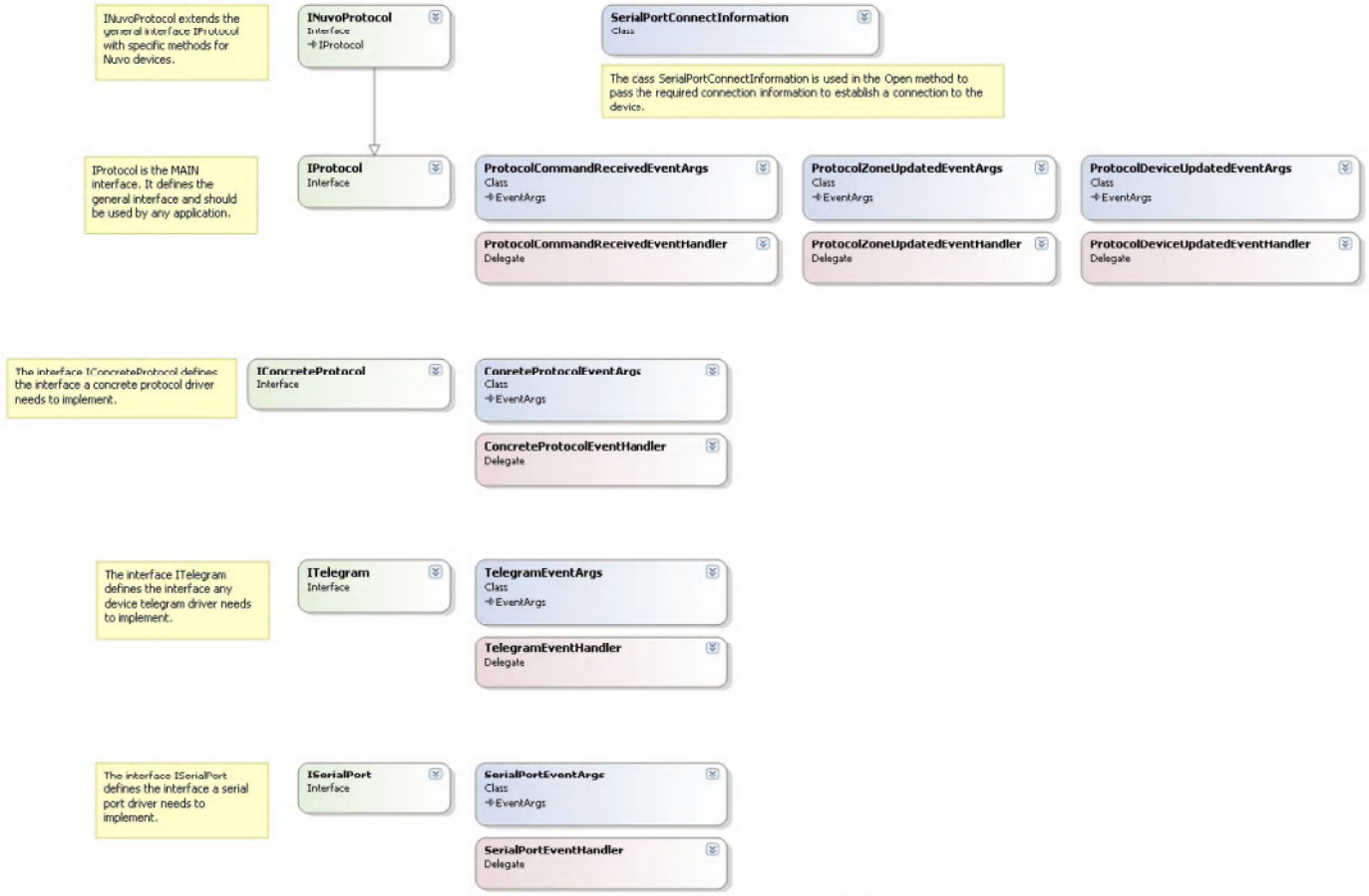


Abbildung 29: Interfaces of the Protocol Driver

The following table describes the interfaces shortly. For a detailed description of the interfaces see the generated help file.

Interface:	Description:
IProtocol	This is the main interface used by the NuvoControl Monitor and Control Service. All systems which would like to be monitored and controlled by NuvoControl are required to implement this interface. There are three events in this interface: • onCommandReceived: is issued in case a command has been received. • onZoneStatusUpdate: is issued in case the status of a zone has changed • onDeviceStatusUpdate: is issued in case the status of a device has changed.
INuvoProtocol	This interface extends the IProtocol interface for specific commands used and supported by NuvoEssentia.
IConcreteProtocol	Public interface, that defines the methods and events which need to be implemented by a concrete protocol class. This interface allows sending and receiving commands. This layer is responsible to create a telegram for a specific command and to parse incoming telegrams to the correct command and its parameters. The event onCommandReceived is issued in case a command has been received.
ITelegram	Public interface, that defines the methods and events which need to be implemented by the telegram layer. This interface allows sending and receiving single telegrams. This layer is responsible to add and remove start and end characters. If available it would also calculate and check an available checksum. The event onTelegramReceived is issued in case a telegram has been received.
ISerialPort	Public interface, that defines the methods and events which need to be implemented by a serial port driver. It's the 'lowest' level interface in the hierarchy. Any serial port driver – or similar drivers, like mock or simulator – which implement this interface can be easily integrated in NuvoControl. The event onDataReceived is issued in case data has been received.

Tabelle 30: Protocol Driver Interfaces

12.1 Interface Events

The following sequence diagram shows a common sequence how spontaneous data is transferred from NuvoEssentia through all layers.

First a connection has to be established between NuvoControl and NuvoEssentia. The required connection information are passed with the *Open* method. If successful and a connection has been established, NuvoEssentia sends changes in the zones spontaneous to NuvoControl.
The serial port class receives these data and sends them with the onDataReceived() event to the telegram layer. The telegram layer parses the content and searches for the start character of a telegram. If found it collects the data until the end character has been received. If a full telegram has been received it is passed with the onTelegramReceived() event to the next layer. The concrete protocol layer is defined with the IConcreteProtocol interface. This class parses the telegram for a valid command and its parameter. If a valid command has been found it is passed to the main protocol layer. This layer is defined with the IProtocol interface.
The interface IConcreteProtocol has device specific commands in its definition. The IProtocol interface is a general non-device specific interface. This allows an abstraction in NuvoControl of device specific features, etc.
The main (or non-device specific) protocol layer supports three events. If a command has been received the onCommandReceived() event is executed. If the zone status has changed the onZoneStatusUpdate() is sent. This allows the client for zone status changes. The client doesn't need to 'think' in commands. The third event is the onDeviceStatusUpdate() (this event is not in the diagram below). This event is sent in case the communication is broken, to indicate the client that no valid connection exists.

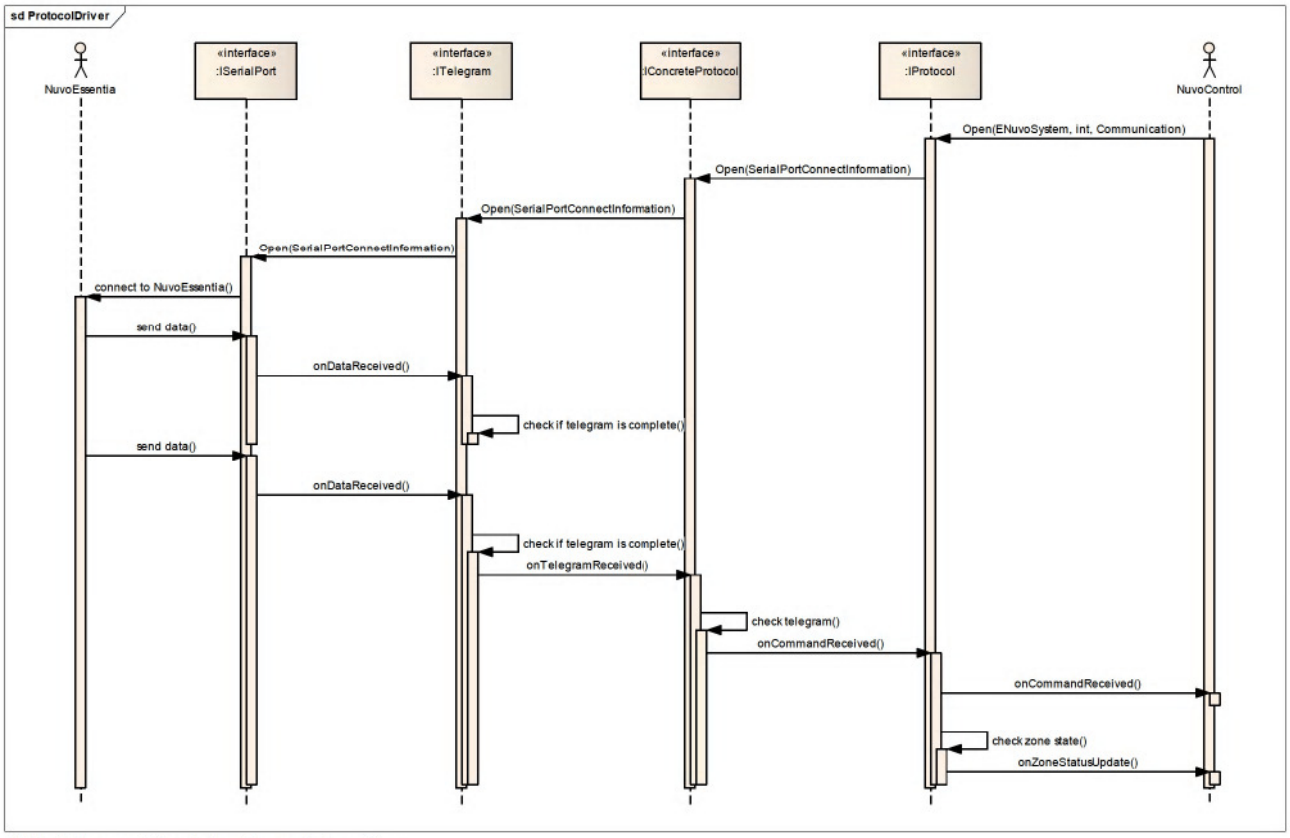


Abbildung 31: Interface Events

12.2 Enumerations

The following enumerations are used in the protocol driver interface.

Abbildung 32: Protocol Driver Enumerations

Enumeration:	Description:
ENuvoSystem	Defines the supported devices (systems) supported by NuvoControl. The current implementation supports only NuvoEssentia.
ENuvoEssentiaCommands	Defines all possible commands of a NuvoEssentia device.
ENuvoTunerT2Commands	Defines all possible commands of a Nuvo Tuner T2 device. These commands are not implemented yet.
ENuvoEssentiaZones	Defines all possible zones of a NuvoEssentia device. Twelve zones are only supported with an additional zone extender amplifier. Without extender only six zones are possible.
ENuvoEssentiaSources	Defines all possible sources of a NuvoEssentia device.
EZonePowerStatus	Defines the possible states a zone can have at a NuvoEssentia device.
EVolumeResetStatus	Defines the possible states the volume reset function of a zone can have at a NuvoEssentia device.
ESourceGroupStatus	Defines the possible states the source group function of a zone can have at a NuvoEssentia device.
EIRCarrierFrequency	Defines the possible IR frequencies a zone can have at a NuvoEssentia device.
EDIPSwitchOverrideStatus	Defines the possible states the DIP switch override of a zone can have at a NuvoEssentia device.

Tabelle 33: Protocol Driver Enumerations

13. NuvoControl.Server.ProtocolDriver.Simulator

The current implementation of the protocol driver contains an in-built simulator. The class ProtocolDriverSimulator implements them. Please note this simulator has nothing to do with the standalone simulator application.

The in-built simulator acts like a serial port driver, it implements the interface ISerialPort. It is started by the NuvoTelegram class, in case the serial port connection information specifies “SIM” as port name. In this case the in-built simulator is started.

The current implementation supports only one simulation mode, called *AllOk*. As the name already implies, the simulator returns on each command request a valid answer that the command has been executed.

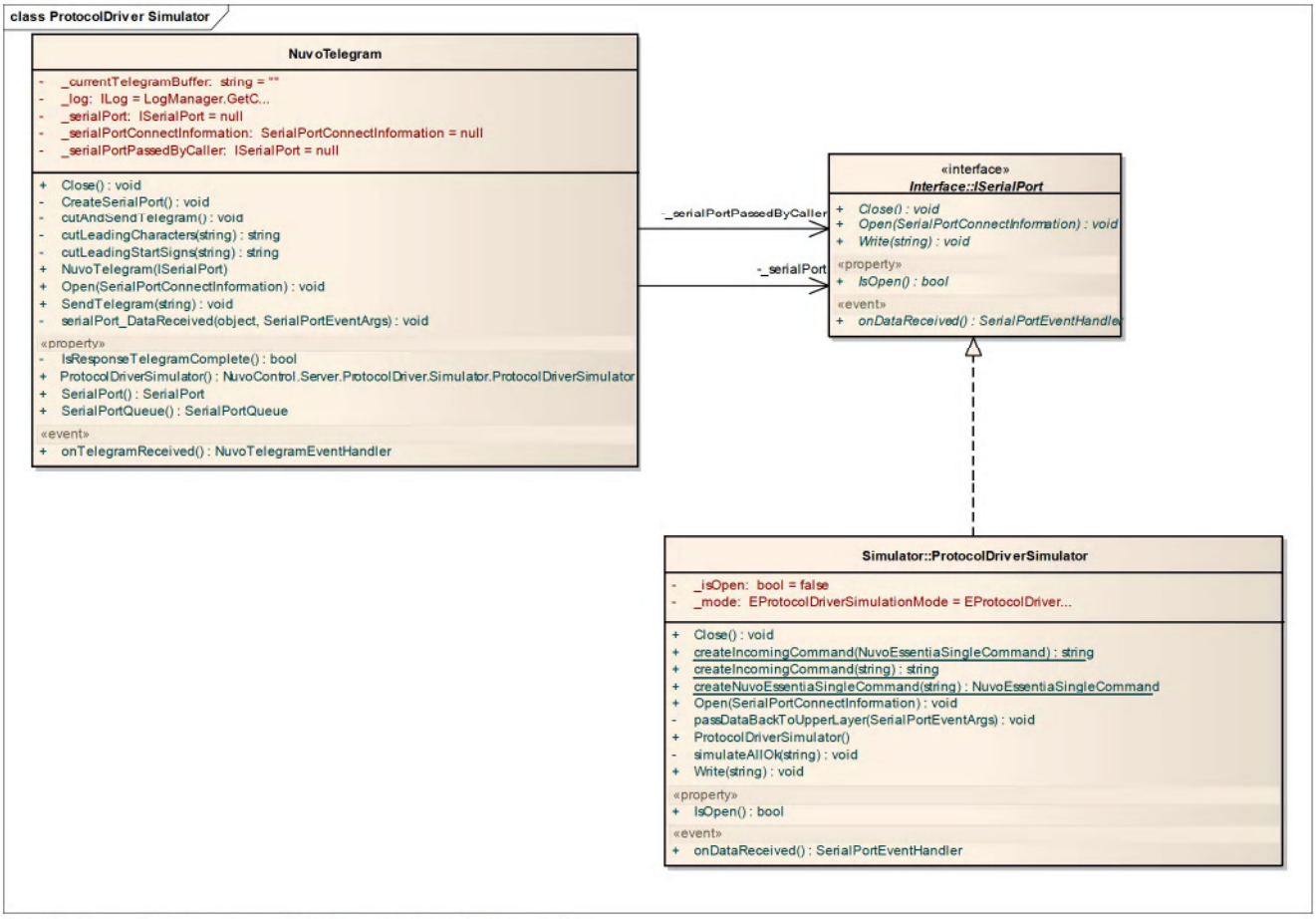


Abbildung 34: Protocol Driver in-built Simulator

14. NuvoControl.Server.WcfHostConsole

This is the WCF host of all server objects. It is responsible for the lifetime of all services and the underlying servers (zone server, function server) as well as the protocol driver.
Furthermore, it specifies the interface endpoints via configuration files. An endpoint is specified by the interface name, the base address (including transportation protocol) and the type of encoding of the interface data.

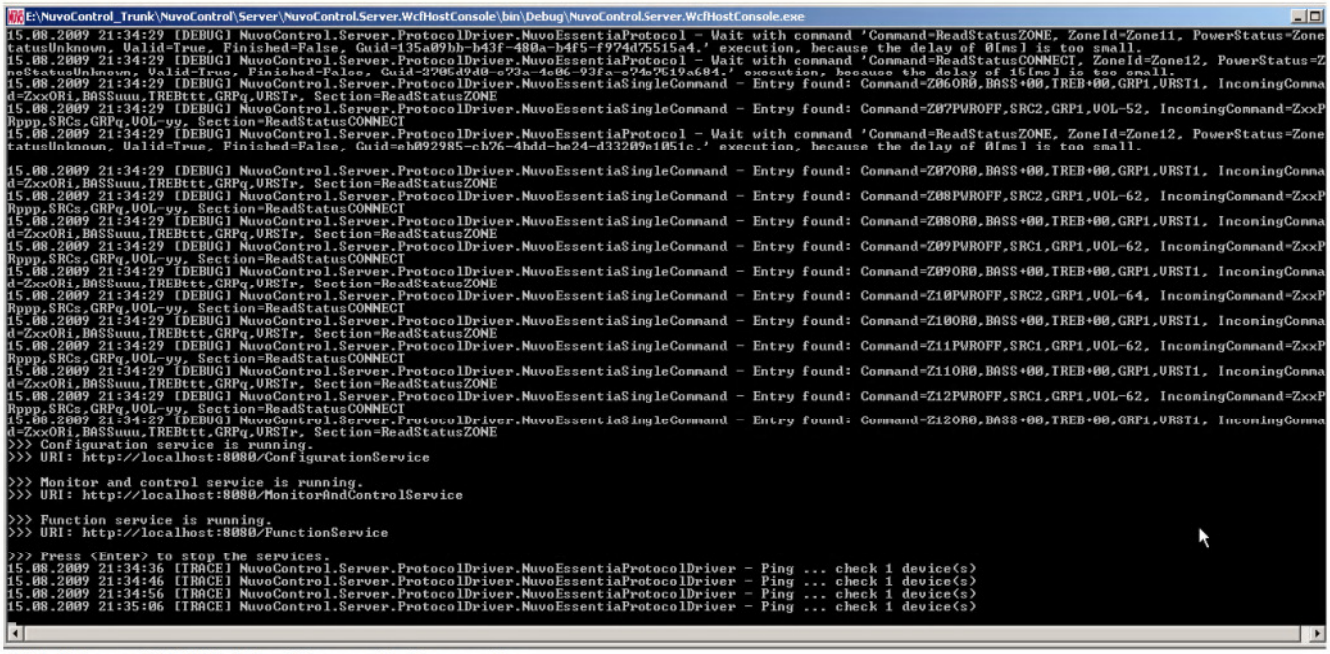


Abbildung 35: WcfHostConsole Console

15. NuvoControl.Server.Simulator

This is the standalone simulator, which has nothing to do with the in-built simulation mode in the protocol driver.
The standalone simulator connects via queues to the NuvoControl system. For this purpose the serial port driver using queues is used. See section 11.3 for more details.

The following picture shows the startup screen of the standalone simulator.
The simulation mode is selected with a drop-down box. If a simulation is running, the progress bar is indicating this. The answer delay defines, how long the simulator shall wait till it answers to an incoming request. The periodic update rate is used by the simulation mode PeriodicUpdate. The text output area shows all incoming and outgoing messages. It shows also simulation mode specific behaviors, like delays etc.
The input message zone control shows always the newest message arrived from NuvoControl.
The four zone controls in the centre are used to edit the simulation data. A change in these controls doesn't generate an outgoing command; it just specifies the data which shall be used in case the zone state is asked by NuvoControl for this zone.
The zone control for spontaneous changes (or also called manual changes), produces spontaneous messages, which are sent immediately to NuvoControl. This zone control simulates the keypads, which can be executed at any (unexpected) time.

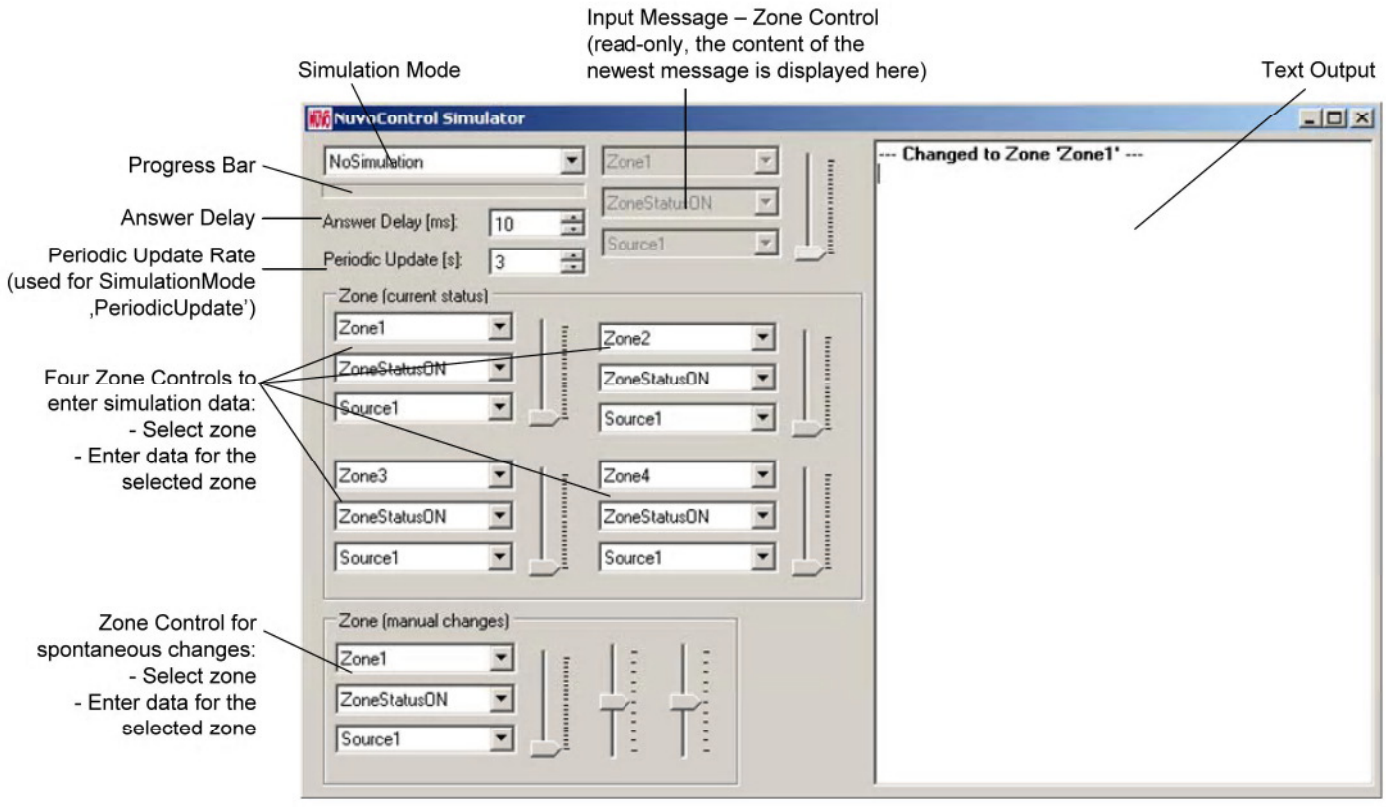


Abbildung 36: Simulator UI

The following simulation modes are supported.

NoSimulation	There is no simulation active. No answer is returned, and no spontaneous events are generated.
AllOk	Simulation is active. Each command is simulated with the correct expected answer. The requested command is parsed and the state is stored in the simulator. An answer is generated to indicate NuvoControl that the command has been executed without any problem.
AllFail	Simulation is active. Return to each command an error message.
WrongAnswer	Simulation is active. On each command is a 'wrong' (but valid) answer returned.
NoAnswer	Simulation is active. No answer is returned on a command.
PeriodicUpdate	Simulation is active. A periodic update is send by the simulator.

Tabelle 37: Simulation Modes

The following picture shows how the simulator and NuvoControl are connected via message queues. If you configure the server to use the serial port driver using the queues, it connects to the two private queues **fromNuvoEssentia** and **toNuvoEssentia**. The simulator connects directly to them.
In the picture is also the MSMQ console, it can be used to view the messages. As example you can see the content of the queue in case of the command *ReadVersion*.

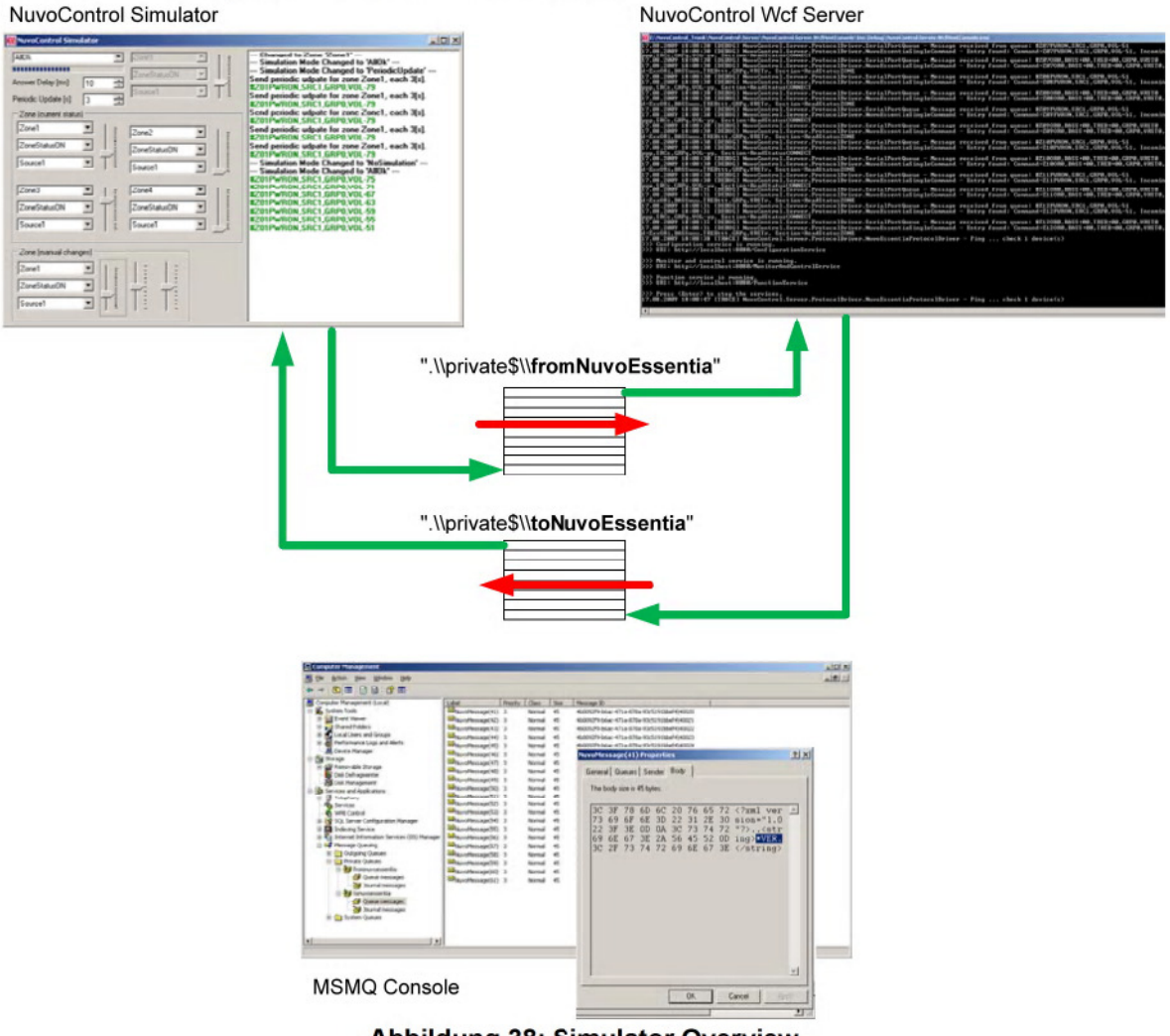


Abbildung 38: Simulator Overview