WWW.JAMK.FI

HYPERNOTIFIER

Piotr Książek

Bachelor's Thesis
December 2009

Degree Programme in Information Technology
Information Technology

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

Author(s)	Type of publication	Date
KSIĄŻEK, Piotr	Bachelor´s Thesis	07122009
	Pages 40	Language English
	Confidential ( ) Until	Permission for web publication (X)

Title

HYPERNOTIFIER

Degree Programme

Information Technology

Tutor(s)

LAPPALAINEN - KAJAN, Tarja

Assigned by

VSoft

Abstract

In every company there is a great amount of information that should be delivered to the employees. Some of them are administrators and they want to know if something was changed in the environment they administrate. If there are many different environments, the employees are getting many different notifications from different sources and in different layouts.

This thesis illustrates a solution that is used to handle different environments to get a plain and compact, unified report about changes that were made. The solution includes collecting of data, storing it and notifying about it. It is a Windows service and a web application using database and interchangeable DLL libraries involved in handling of the specific sources of changes. The solution provides a simple mechanism for the creation and maintenance of reports containing selected changes from selected sources supplied at specified time and frequency.

The Hypernotifier solution is created using C# programming language, and it runs in the Microsoft® .NET Framework. It also takes advantage of the existing software infrastructure.
The solution was made for the employees of VSoft, who would like to be notified about changes in some areas of their interests, for example a folder in file system or a list at the SharePoint web page.

Keywords

Changes, ASP.NET, .NET, services, distributed applications

Miscellaneous

Table of contents

Figures and tables

TERMINOLOGY

C#

Multi-paradigm programming language encompassing imperative, functional, generic, object-oriented (class-based), and component-oriented programming language. It was developed by Microsoft within the .NET initiative. C# is one of the programming languages designed for the Common Language Infrastructure.

ASP.NET

Web application framework developed and marketed by Microsoft to allow programmers to build dynamic web sites, web applications and web services.

SQL

Structured Query Language is a database computer language designed for managing data in relational database management systems (RDBMS). Its scope includes data query and update, schema creation and modification, and data access control.

ADO.NET Entity Framework

Entity Framework is an object relational-mapping (ORM) framework for the .NET Framework. This framework is an ORM offering from Microsoft for the .NET Framework. While Microsoft provided objects to manage the Object-relational impedance mismatch (such as a DataSet).

ADO.NET Entity Framework is included with .NET Framework 3.5 Service Pack 1 and Visual Studio 2008 Service Pack 1, released on 11 Aug 2008. It also includes the capability of executing LINQ against ADO.NET Entity Framework entities.

.NET Remoting

Is a Microsoft application programming interface (API) for inter process communication.

1 INTRODUCTION

This thesis will demonstrates the development of a Hypernotifier solution, one that is able to subscribe in order to receive selected business information simultaneously from multiple sources of data as opposed to the currently available mechanisms in a company for juveniles "technical sign up on a single list / RSS's / source", and process them as required.

The technologies that are involved with this solution, and the environment, into which it will be created are looked into in the thesis. The study then proceeds to examine the actual implementation of the solution.

1.1 The need for a notifying application

In a company like VSoft there are many sources of information that employees would like to get. There are many different types of sources with different collections of information. Employees and especially administrators would like to be informed if something somewhere has been changed. And if they are overwhelmed with many different reports, it is hard to get the information that they are really interested in. It is also hard to configure the notifications according to the needs of layout, the accuracy of the information and time of reporting. So the employees would like to be informed about the changes in different sources of data in the same, easy form for each and in a fast and easy configurable way.

Hypernotifier is a tool for collecting and reporting any changes in data sources. The architecture provides solutions to create any number of plugins which collect data from the subsequent data sources. Changes are reported in the user-selected areas of interest, in a clear form. They are delivered in due time and with appropriate frequency, and only including the data to which the user has read permissions. The collected data are available independently, from the sources of data, which provides quick and secure access, and may include links that can redirect to the details directly in the data source. The report can include data since the last report or generate a selected period of time in terms of available data in the database of Hypernotifier.

1.2 VSoft JSC

VSoft Ltd is a provider of advanced IT solutions in the area. It creates information systems and serves customers optimizing complex business operations.

The roots of the company date back to 1996. From the existing number of dynamically developing companies, as a result of a merger has been established one innovative and resilient company - VSoft Limited Company. The Company has exclusively Polish capital. The purpose of the merger was the concentration of knowledge, technology and capital, so that it provided clients with the highest quality services. The developed systems are tailored to meet the individual needs of the clients. The company works with leaders in their industries. In 1999, cooperation was established with PKO Bank Polski, in 2001 - with Gbg Ltd, in 2004 - with PZU Ltd and BIK Ltd. The company cooperates with the biggest Polish companies.

The company specializes in the design and construction of modern solutions in the field of information technology and data. It offers a professional service and is a reliable partner in the development and improvement of large areas of the business activities of the clients.

VSoft Ltd has reached an extremely strong position as a supplier of complete and effective solutions. The adopted philosophy of innovative, unconventional, but, above all, effective action is accomplished due to many years of experience, supported by a number of successful implementations.

VSoft creates custom solutions for clients who require value-added service in the form of optimizing business and operational efficiency. This competence is dedicated to Microsoft partners for creating a solution using Microsoft Visual Studio 2005, Visual Studio Tools for Office, SQL Server, Windows Server 2003, BizTalk, Windows XP and Office System 2003.

VSoft company can boast certification from Microsoft: Custom Development Solutions, Microsoft Gold Certified Partner, ISV Software Solutions, Business Process and Integration Solutions.

(www.vsoft.pl)

2 DESIGNING THE SOLUTION

2.1 The deployment environment

The whole system is located on one server for easier communication between its parts. Therefore, there is the main application, plugins as DLL libraries and the web application. On the same server there is a database so it is also a database server.

2.2 Requirements

Before the planning or implementation phases of development, the first step is to gather information about the requirements that the tool needs to provide.

2.2.1 Functional requirements

Use case

Based on the assessment of the problem at hand, the following use cases were identified for the solution. FIGURE 1 presents the use case diagram.

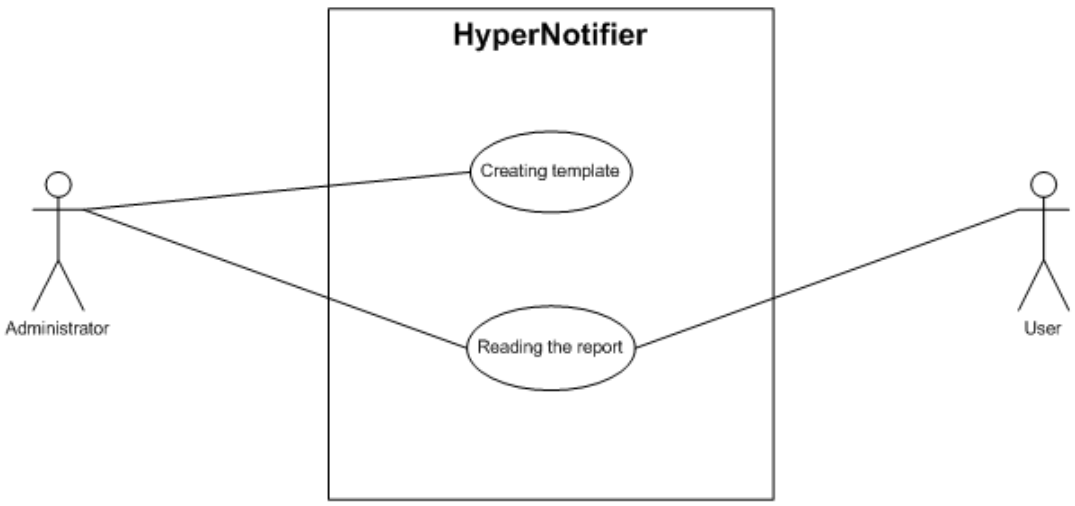

FIGURE 1. Hypernotifier use case diagram

Figure 1 presents two types of user profiles: Administrator and User. The first one is able to configure templates, while the second one can only receive and view them filled with data.

The use case diagram depicts the Hypernotifier usage from the user's point of view. Two actors that interact with the system can be seen.

The requirements for the solution are:

1. To build a general platform to collect and publish any changes to the data source
2. To build plug-ins at least for the file system, SharePoint 3.0, TFS SourceControl, TFS WorkItems.
3. To be able to completely replace the current system of notifications on inner company portal.
4. To implement, as far as possible, plug-ins to: VBP, TFS Reports.
5. To improve corporate communications.
6. To give proper information at the right time.
7. To streamline the recipient by providing a number of summary reports instead of a well-defined number of notifications for the every current single change.
8. To increase the positive image of the inner corporate portal, and encourage the publication of the proposal and other changes.

Fields tree

A fields tree shows a list of available fields for a specified data source. The user can choose them by checking the appropriate checkbox to specify which information he would like to get and in which position.

Constraints fields

Constraints fields are fields where users decides how they would like to information be chosen from gained data. From which fields and how constraints will be created.

Templates chooser

The user can choose the stored templates and modify them afterwards as he needs.

Report creator

Report creator is a part of the web site where users decide when the report will be send, how often and who will get it.

Save

A template can be saved to the database with additional data like e.g. sending conditions.

2.2.2 User requirements

The following requirements were analyzed and identified while designing the solution.

TABLE 1. User Requirements.

ID	Requirement description	Reference
1	*Receiving information about changes* The solution has to be able to receive incoming information about changes.	2.2.3 1.b
2	*Scanning source for changes information* If source cannot send notification that something was changed, tool should ask for the required information or get it somehow from the source.	2.2.3 1.a
3	*Understand different kinds of data sources* Handle the differences in data sources.	2.2.3 2
4	*Storing changes information* Information about changes needs to be stored.	2.2.3 4
5	*Creating templates* The ability to create different kinds of new templates which could be used to present information about changes.	2.2.3 5
6	*Filling templates* Templates are connected with queries that will be fired to fill the fields of the template.	2.2.3 5
4	*Sending reports* The solution should have ability to send mails or different kinds of notification.	2.2.3 6
5	*Integrating and managing information* Using previously available information, integrating it, and managing new information regarding notification.	2.2.3 4
6	*Secure messaging* Users should be able to see only the information to which they have permissions in the data source.	2.2.3 3

2.2.3 Notification processing functionality

After summarizing some of the functional requirements in the previous section, here the details for the functionality to be provided are discussed in more detail. At this point, only a textual description of the functionality is given.

1. Getting information from data source
   It should be done in two ways:
   a. Active mode – the data source is unable to send notifications that something was changed, so the plug-in should watch the source and get the required information. It can be done by continuously asking the source for data and comparing it with the stored data. This mode is obligatory meaning that every plug-in should provide such functionality.
   b. Passive mode – when the data source has ability to notify that something was changed then the plug-in should only handle the received data. This is an optional mode because some sources do not have ability to react to an inner change.
2. Data normalization – plug-in should know what data are stored in the database and how to handle and present them because the Hypernotifier service does not care what is stored as a change.
3. Scanning permissions – to know if users can get information about specific part of data source. Only the plug-in knows what the source of the change is, and because there are differences in the way of storing permissions in different sources only the plug-in can retrieve and handle the information about permission to viewing the change.
4. Storing data – Information about changes will be stored in database as xmls and metadata. One change stored in one row of table will be connected with many, different metadata stored in another table. Metadata is needed to easily find a change that is required. It contains common data that is used to query for the change.
5. Creating a template – the administrator will be able to use stored templates to create reports for users or can create his/her own template which will be filled during the sending message process. Creating a template will be connected with creating a query to the database which afterwards will be used to fill the

template to create a report. The templates and queries will also be stored in the database.

The query can ask about:

- Who made the change? (filter for persons + “all” + “me”)
- What was changed? (data types filter + “all”)
- When has the change been made? (data range + “all”)
- Where was the change? (localization and source filter + “all”)
- How to present the change? (template type selection)

6. Sending a report – filled reports can be sent to the user differently. The most common way is sending the mail with a filled report. It can be determined when and to whom the mail should be sent.

2.2.4 Non-functional requirements

Security

The application cares all the time for the security of the data. Every moment that the user has access to data before showing it the application checks if he has appropriate permissions.

Dictionary

All possible texts in the application should be taken from dictionary. The application has standard texts which can be used in many parts of it so keeping them in a database is a good idea; moreover, it gives a readymade place for managing the texts which can be used to prepare translations for different languages.

2.2.5 Constraints

Validation

Before saving the template the minimum amount of data must be given by the user. It is at least one field to show on a report, at least one user as receiver of the report and the data needed to send an e-mail.

2.3 Database design

Next, the database format is introduced. The database that the Hypernotifier will rely on provides some features that will aid in the construction and processing of the report.

Many of the required information are already readily available in the databases that are in use at the company currently. However, they need to be integrated, using the messaging solution, because often the related information is in no connection in the database.

Before exploring the table designs for the solution, it is a good idea to look at a list of information that we will be working with. This is the information that has to be considered storing in the database.

- Change information (user, name, date, change type, etc.)
- User information (receivers, administrators)
- Management information (for example, templates or queries)

Given these objects that should be stored, it is possible to arrange the information into database tables.

2.3.1 Change storage

The change information is stored in database in a XML format which completely describes it. In addition to this, an extract is created from the change that summarizes what kind of change it is, when it happened and who has done it.

The purpose of METADATA table is to allow fast access to the most common fields of a change. The whole information about a change is kept in CHANGES table, and can be accessed through this table, when needed.

CHANGES

	Column Name	Condensed Type	Length	Nullable
🔑	ID	int	4	No
	CHANGE_XML	xml	-1	Yes
	PLUGIN_TYPE	nvarchar(10)	10	Yes

FIGURE 2. CHANGES table structure

METADATA

	Column Name	Condensed Type	Length	Nullable
🔑	ID	int	4	No
	CHANGE_ID	int	4	Yes
	ATTR_TYPE	nvarchar(10)	10	Yes
	ATTR_VALUE	nvarchar(50)	50	Yes

FIGURE 3. METADATA table structure

The METADATA is a table that saves the common data for the change that can be easily used to query about the change. To make a query fast the change will be looked up by the metadata.

2.3.2 QUERIES

The queries made by the administrator also have to be stored because they will be used during process of filling and sending a report about the changes. The purpose for this table is to remember fields selected to be filled and how they should be filled.

QUERIES

	Column Name	Condensed Type	Length	Nullable
🔑	ID	int	4	No
	QUERY	xml	-1	Yes
	TYPE	varchar(10)	10	Yes
	CREATED	datetime	8	Yes
	[USER]	int	4	Yes
	SEND_TIME	nvarchar(MAX)	-1	Yes
	NEXT_SEND_TIME	datetime	8	Yes

FIGURE 4. Address book table structure

2.3.3 DICTIONARIES

In almost every application there are some types of elements that can be categorized and reused many times; this is a good reason to create dictionaries. The table DICTIONARIES contains types of collections that will be used in different parts of the application. This table is connected with ELEMENTS, which saves the types that appear in the application.

DICTIONARIES

	Column Name	Condensed Type	Length	Nullable
🔑	ID	smallint	2	No
	NAME	nvarchar(20)	20	Yes
	GUID	uniqueidentifier	16	Yes

FIGURE 5. DICTIONARIES table structure

DICT_ELEMENTS

	Column Name	Condensed Type	Length	Nullable
🔑	ID	int	4	No
	DICT_ID	smallint	2	Yes
	NAME	nvarchar(20)	20	Yes
	GUID	uniqueidentifier	16	Yes

FIGURE 6. DICT_ELEMENTS table structure

2.3.4 Database structure

For storing the data in the application, a new database was created. Although some of the information was already present in the database, it was definitely the time to start gathering EDI messaging related information into one central place.

The previous sections show the individual structure of the database tables used. They used C# data types, and provided no indication on the actual relation of the tables.

FIGURE 7 shows all the tables in the database, including their structure, and the connection between the tables.

This image illustrates in detail the data types used for the different fields of the database, as well as the primary key columns of the tables.

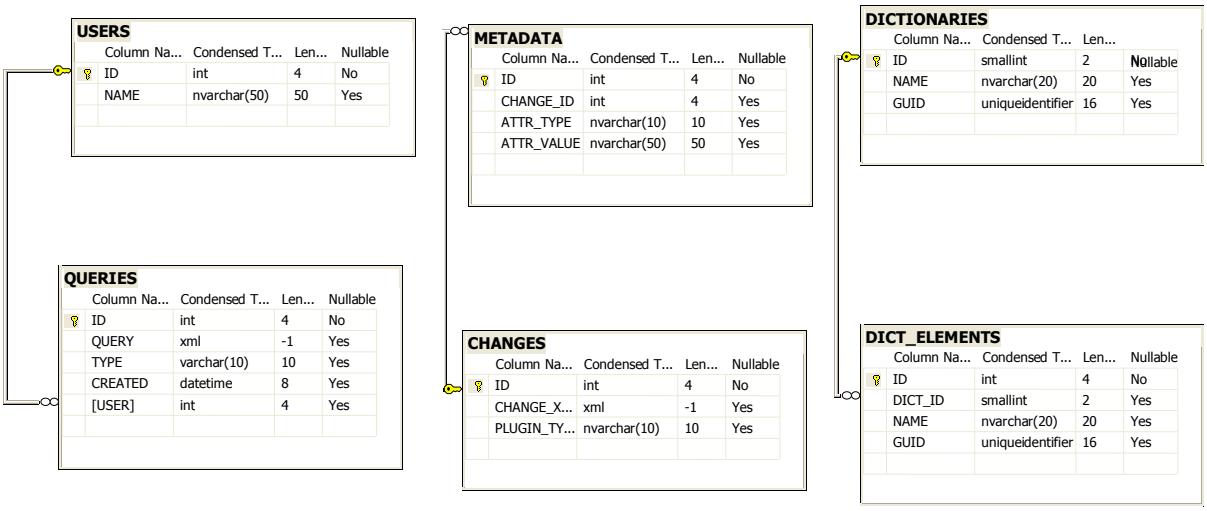

FIGURE 7. Database structure

3 IMPLEMENTATION OF THE NOTIFYING SOLUTION

This chapter will explore the implementation of the notification solution. First, the architecture of the entire application is introduced followed by the details of each part and the information flow through all the parts and interaction with the end user are shown.

3.1 Hypernotifier architecture.

Before detailing the different parts of the Hypernotifier architecture, an overview graphic that depicts how the system is organized is illustrated. The different parts and applications presented in **Błąd! Nie można odnaleźć źródła odwołania.** will be explained and explored further in this chapter.

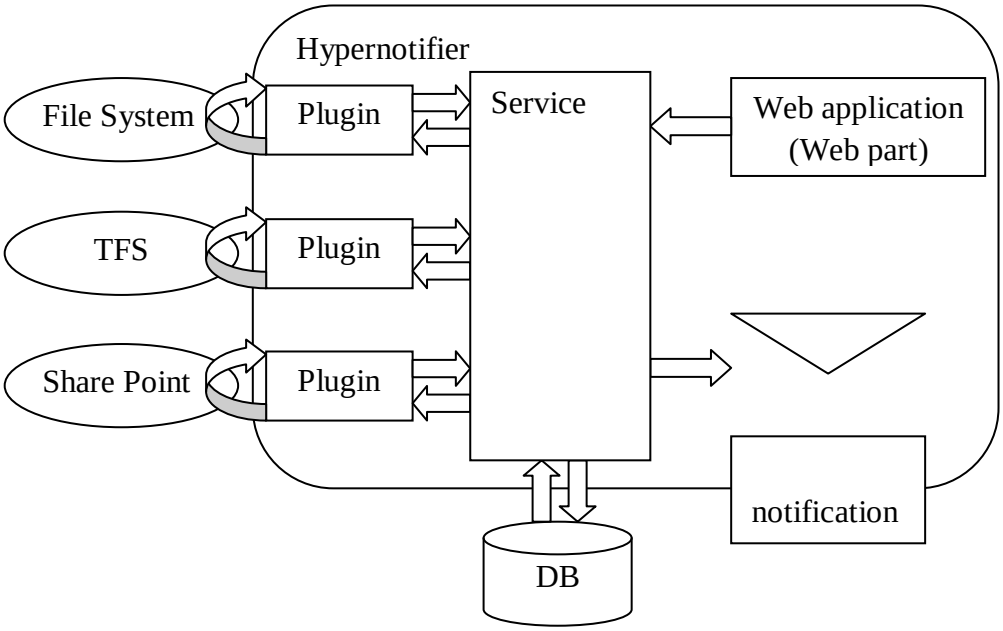

FIGURE 8. Hypernotifier architecture.

As far as the different parts of this chapter are concerned, the reader might want to refer back to this image, to see how things are connected.

The three ellipses on the left represent external systems from which information is extracted. Hypernotifier generally consists of three parts: plugins, service, web application; and is connected with database.

Before further exploring the systems that make up the notifying solution, more fundamental building blocks are discussed in further detail.

3.2 Interfaces

In the Hypernotifier solution there are two projects which consist only of interfaces that are: IPlugin and IHyperNotifier as shown on FIGURE 9.

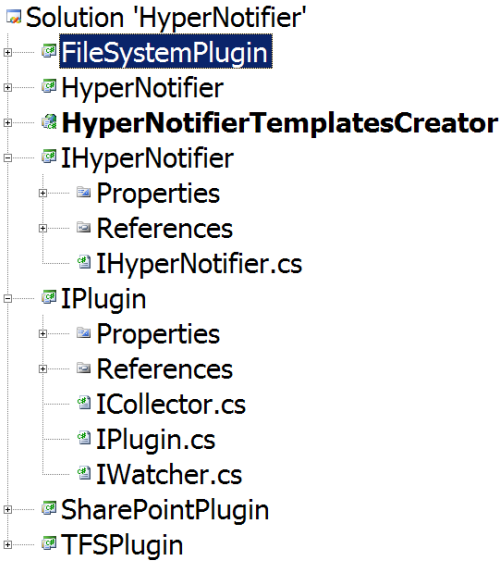

FIGURE 9. HyperNotifier solution.

Interfaces are created in separated projects because they are used for .NET Remoting communication that will be further explained in more details.

3.2.1 IPlugin

IPlugin interface describes the functionality of the plugins:

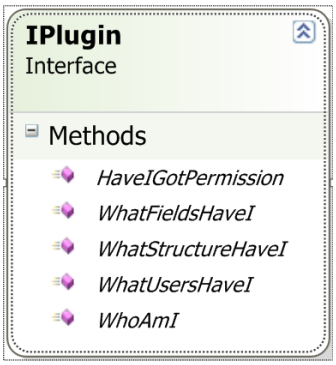

FIGURE 10. IPlugin interface.

- Method *HaveIGotPermission* checks if a given user has permission to read data about changes in a given path.
- *WhatFieldsHaveI* returns names of fields that can be put into information about a change in a particular data source.
- *WhatStructureHaveI* tells how information is located in the data source (eg. in FileSystem it returns the folders' structure by passing all the paths that can be reached).
- *WhatUsersHaveI* returns the list of all users who are using precisely this data source.
- *WhoAmI* gives the name of the data source.

3.2.2 ICollector

ICollector is implemented by plugins which only receive information about a change sent by the data source.

3.2.3 IWatcher

IWatcher interface indicates that the plugin which implements it is observing the data source and gaining information by itself.

3.2.4 IHyperNotifier

IHyperNotifier interface tells what a service can do.

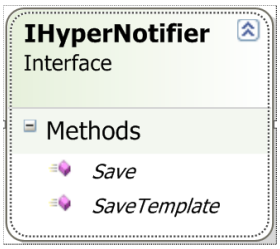

FIGURE 11. IHyperNotifier interface.

- *Save* – saves the data about the change to the database.
- *SaveTemplate* – saves query and sends the time to the database.

3.3 Plugins

Plugin is a part of Hypernotifier which is responsible for the whole communication between the source of data and service.

Plugin is connected to the project as DLL that stands for **Dynamic-link library**. There can be an unlimited amount of plugins, each of them cares for one data source.

Plugins can receive information about changes in a data source and just pass it to the service or observe the source and gain information by itself. In the first case a plugin has a class which implements ICollector interface from IPlugin, while the class which watches for changes in the data source implementing IWatcher. A plugin can simultaneously have both of these interfaces implemented and get information in both ways.

Every plugin implements IPlugin interface which ensures that the plugin provides all the additional information about data source like its structure and a list of users. The next important thing is that a plugin always ensures the security of information by checking if a user has permission to read the data which wants.

3.4 Service

This is the main part of the application, in this solution it is a project called HyperNotifier because here all the work of the application is done. Because the tasks

involving the duties of HyperNotifier are mostly background processing tasks and because it must watch all the time for changes and regularly send notifications the implementation was decided to be a Windows Service.

During the implementation phase the problem arose that communication between different parts of solution like service, DLLs and web application was needed. There were many solutions planned, and finally it was decided to implement .NET Remoting with IPC (**Inter-process communication**) as communication channel (for details on .NET remoting, see Section) because the application was planned to stand on one server.

3.4.1 Windows Services with C#

In contemporary C# programming language using the .NET framework the creation of Windows is very easy, comparing to how Windows services are created and registered in such languages as C or C++ using the WIN32 API.

Microsoft Windows services, as said in Introduction to Windows Service Applications (2009), known as NT services, enable users to create long-running executable applications that run in their own Windows sessions. These services can be automatically started when the computer boots, they can be paused and restarted, and do not show any user interface. This makes these services ideal for the use on a server or whenever there is a need of long-running functionality that does not interfere with other users who are working on the same computer. The services can also be run in the security context of a specific user account that is different from the logged-on user or the default computer account.

Creating a service is creating a Microsoft Visual Studio .NET project, defining code within it that controls what commands can be sent to the service and what actions should be taken when those commands are received. Commands that can be sent to a service include starting, pausing, resuming, and stopping the service, and executing custom commands. (http://msdn.microsoft.com/en-us/library/d56de412%28VS.80%29.aspx. Referred to on April, 2009)

To create a Windows Service Application *System.ServiceProcess.ServiceBase* which is provided by .NET framework should be derived in service base class, and this class is the base class for every service written in the .NET framework. There are two methods that need to be overridden for a service to become functional. These methods are *OnStart* and *OnStop*. All the low level details of creating and using the service environment are handled by the .NET framework.

After creating and building the application, users can install it by running the command line utility **InstallUtil.exe** and passing the path to the service's executable file, or by using Visual Studio's deployment features. Users can then use the Services Control Manager to start, stop, pause, resume, and configure the service. Users can also accomplish many of these same tasks in the Services node in Server Explorer or by using the **ServiceController** class.

Visual Studio .NET which was used to create HyperNotifier, ships installation components that can install resources associated with service applications. The installation components register an individual service on the system to which it is being installed and let the Services Control Manager know that the service exists. When working with a service application, user can select a link in the Properties window to automatically add the appropriate installers to the project.

When adding an installer to the project in Visual Studio .NET, a new class (which is named **ProjectInstaller**) is created in the project, and instances of the appropriate installation components are created within it. This class acts as a central point for all of the installation components that a project needs. For example, if users add a second service to the application and click the Add Installer link, a second installer class is not created; instead, the necessary additional installation component for the second service is added to the existing class.

(http://msdn.microsoft.com/en-us/library/d56de412%28VS.80%29.aspx. Referred to on April, 2009)

3.4.2 HyperNotifier

The service which works in our solution can be divided into two parts from the business point of view. One part is responsible for gaining and saving data into database, and the other one is creating and sending the required notifications.

Gathering information is done by a plugin and there it is called by a service method which saves it into the database. This method is implemented in class SaveBLL as well as the method SaveTemplate which saves the query and sends information created by the user on a web site.

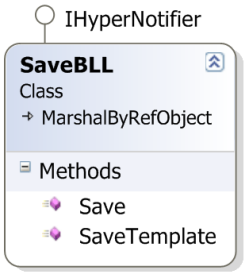

FIGURE 12. SaveBLL class.

The next important class is Query class which cares for processing the queries created by the user. Here the author would like to point out that every data in the application like changes in the data, queries and the sent information are passed as XML data because this is the easiest way to store them in the database. This is the reason for creating a class which can understand and use XMLs passed through the application.

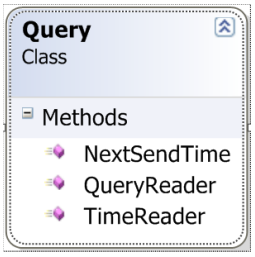

FIGURE 13. Query class.

In this class the proper information from the database is obtained which is used to create a report about the required changes. This information is collected when the timer in the service reaches the next sending date set by the user. Then all the data are placed into the template with the help of XSLT and they are sent to the listed users as a ready report.

Service is also some kind of link between a plugin and a web application because each of them does not know about the other, they only communicate with the service and it manages the flowing information. Additionally, only the service has access to the database so both plugin and web application just pass the data to the service and it cares for saving them to database and if needed, retrieves and passes the data further. Thus if in plugin or web application paragraphs saving or retrieving data from database were discussed, it is thinkable that the data are passed through the service, not directly to the database.

3.5 Web application

This part of the solution is nearest to the user and here is all the communication between the application and the user. The name of this web application is HyperNotifierTemplatesCreator because it is mostly responsible for giving the functionality for creating and saving queries as templates for future filing. This part was required to be as simple as possible so it is one ASP.NET site which is clear and looks like a simple form to fill. This web site can be accessed directly and also is prepared to be a web part which can be put on another web site especially in SharePoint environment.

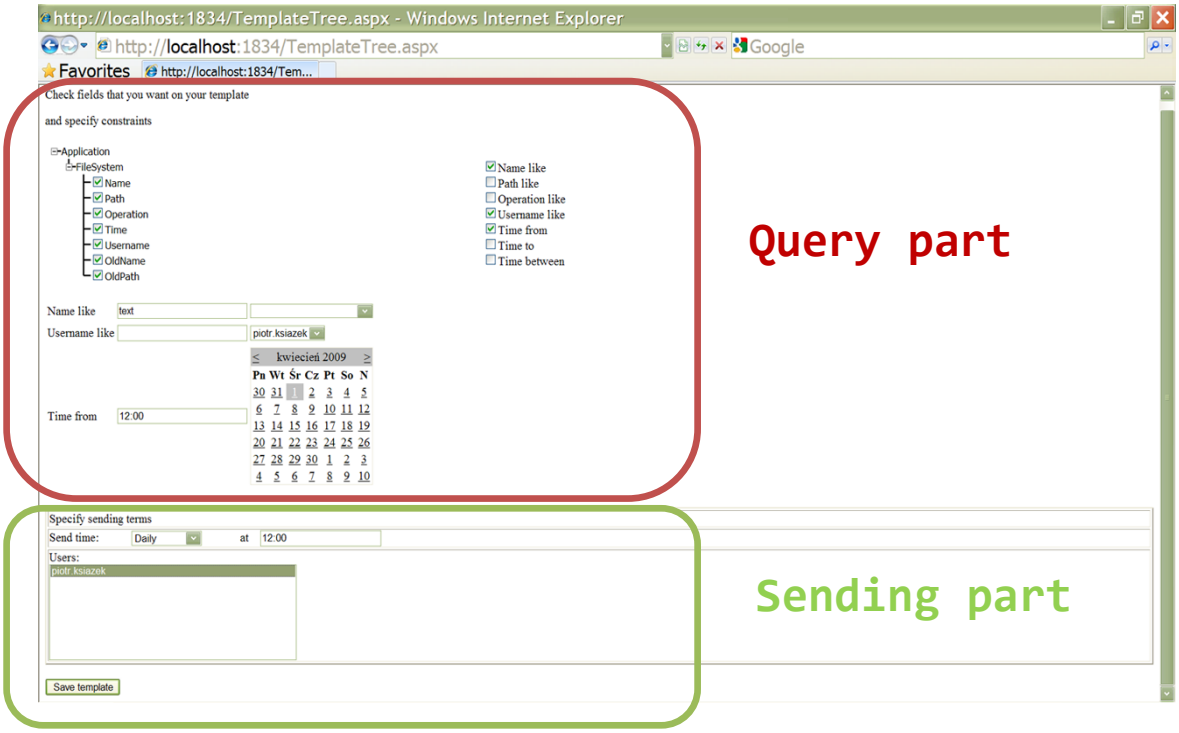

FIGURE 14. Example of HyperNotifier web site look.

From a business point of view the page can be divided into two parts as shown in FIGURE 14. The *Query part* is responsible for creating a query which means selection of fields that are on the report, and specification of the constraints which will be used for getting information. Meanwhile, the sending part allows the user to decide when, how often and to whom notification the report will be sent. Next, both parts are discussed briefly.

3.5.1 Query part

During loading of the page there are a number of issues done to prepare both of the parts for users. *Query part* has some controls which are filled before users see them:

- Field tree

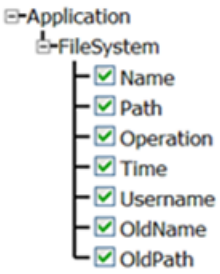

FIGURE 15. Field tree

All accessible data sources are presented here with all the fields that can be put on template.

- Constraint fieds

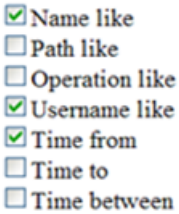

FIGURE 16. Constraint fields

On the right of *Field tree* is shown a list of check boxes which are used to check which fields are to be restricted. They are shown for each data source independently. After checking the selected fields, under the *Field tree*

accordingly appear controls, which allows imposing restrictions at the selected fields.

- Constraint controls
  These controls can be filled by writing a constraint or by selecting it from a given list which is filled with data retrieved from the data source. The user can also write some filter which allows limiting the list of constraints. The important issue is that if the user does not have permissions to see some information, he will not see it because all the data are checked for permissions before displayed on the list.

3.5.2 Sending part

This part is used to decide how often the report will be sent and who will receive it.

Therefore, controls allow a user to decide if a report must be sent daily, weekly or monthly, at which time or even on a specified day of the week. Reports can also be sent immediately after the required change appears in the data source.

In this part is also the list of users who will get the report. This is the list of all the users who are using the source application and also a list of groups in the application if the report should be sent only to a specific group. So users can be choosen by name or just the group picked.

After selecting all the above information the user can press the button which is at the bottom of the page called *Save template*. This action causes that the application gets all the selected information, prepares appropriate XMLs and saves the data into database.

4 USED TECHNOLOGIES

4.1 The .NET framework and ASP.NET

This section introduces the application development process using the .NET Framework.

There are many application platforms, and a great amount of them are platforms for web applications. Because of bigger accessibility, the user interaction side of the Hypernotifier is based on ASP.NET.

ASP.NET is the web application platform provided by Microsoft, and it is run on the Windows Server operating system, using the Internet Information Services (IIS) web server and the .NET Framework.

4.1.1 The .NET Framework

The **Microsoft .NET Framework** is a software framework that can be installed and used on computers running Microsoft Windows operating systems. It includes a large amount of libraries containing coded solutions for common programming problems and a managed environment in which programs written specifically for the framework can be launched.

Base Class Library included in the framework provides a large range of features for common programming needs like user interfaces, data and data access, database connectivity, cryptography, web application development, numeric algorithms, and network communications. (http://en.wikipedia.org/wiki/.NET_Framework. Referred to on May, 2009) This class library is used in programmers' own code mostly for basic operations, and in combination with their code it makes up applications.

The software environment in which the programs operate, manages its runtime requirements. This runtime environment is known as the Common Language Runtime (CLR). The CLR works as an intermediate layer between the launched program and CPU that will execute the program so that programmers need not consider the

capabilities of hardware environment. The CLR also provides more utilities, such as security, memory management, and exception handling. The class library and the CLR are the core of the .NET Framework.

The previous version of the .NET Framework which is 3.0 is included in Windows Server 2008 and Windows Vista. The current version 3.5 can also be installed on Windows XP and the Windows Server 2003 operating systems. On Windows Mobile platforms and smart phones there is available a reduced version of the .NET Framework called .NET Compact Framework. Version 4.0 of the framework was released as a public Beta on 20 May 2009.

Basic architecture of the .NET Framework is as follows:

Common Language Infrastructure (CLI)

This provides a language-neutral platform for application development and execution, including solutions for exception handling, garbage collection, security, and interoperability. Because the implementation of the core aspects of the .NET Framework is done according to CLI, it can be used across the many languages supported by the framework and is not limited to a single language. Microsoft's implementation of the CLI is called the Common Language Runtime, or CLR. (http://en.wikipedia.org/wiki/Common_Language_Infrastructure, Referred to on May, 2009)

Assemblies

The Common Intermediate Language to which .NET Framework languages are compiled is stored in .NET assemblies. Assemblies are stored in the Portable Executable (PE) format that is used in the Windows platform in DLL and EXE files. The assembly can be one or more files, one of them must contain the manifest, which has the metadata for the assembly. The complete name of an assembly is a simple text name, version number, culture, and public key token and should not be confused with the filename on disk. The public key token generated when the assembly is compiled is a unique hash, which guarantees that two assemblies with the same public key token, are identical for the framework. Also a private key can be specified which is required to add an assembly to the Global Assembly Cache. The private key is used for strong naming and it proves that the assembly has the same author when a new

version of the assembly is compiled because it is known only to the creator of the assembly.

Metadata

.NET metadata describes all CIL that means all classes and class members that are defined in the assembly, and also the classes and class members that are connected with the current assembly. The CLR checks the metadata to ensure that the correct method is called. The metadata can be created by developers through custom attributes but usually is generated by language compilers. The metadata contains information about the assembly as written above, and is also used by the reflection functionality of the .NET Framework.

Security

.NET security mechanism consists of following features: Code Access Security (CAS) and validation and verification. Code Access Security is a solution that prevents code without appropriate permissions from performing privileged actions. It is based on evidence (that commonly is the source of the assembly, either local machine or Internet) that is associated with a specific assembly. Code Access Security uses evidence to determine the permissions that code possess. If some code is called by another code it can demand from the caller to have a specific permission. This demand causes the CLR to perform a check through every assembly of each method in the call stack for the required permission; if any of assemblies has not required permission a security exception is thrown.

Class library

The .NET Framework is shipped with a set of standard class libraries. Each class library is organized in a hierarchy of namespaces. The most commonly used classes are contained in either `System.*` or `Microsoft.*` namespaces. A large number of common functions is implemented there, such as input/output features, graphic rendering, database interaction, and XML document manipulation, among others. The .NET class libraries are not restricted to one language but are available to all .NET languages. The .NET Framework class library can be divided into two parts: the Base Class Library and the Framework Class Library.

The **Base Class Library** (BCL) is the core of classes that serve as the basic API of the Common Language Runtime. The classes in `mscorlib.dll` and some of the classes in `System.dll` and `System.core.dll` are a part of the BCL. The BCL classes are available in every .NET Framework implementations including .NET Compact Framework, Microsoft Silverlight and Mono.

The **Framework Class Library** (FCL) refers to the entire class library that .NET Framework includes. It contains large set of libraries, including WinForms, ADO.NET, ASP.NET, LINQ, Windows Presentation Foundation, Windows Communication Foundation among others. The FCL has a comparable range to the standard libraries of Java and much greater range than standard libraries for languages like C++.

Memory management

The .NET Framework CLR frees the developer from thinking about memory (allocating and freeing up when done); instead it does the memory management itself. To this end, the memory allocated to instantiations of .NET types (objects) is done contiguously from the managed heap, a pool of memory managed by the CLR. ("Garbage Collection: Automatic Memory Management in the Microsoft .NET Framework". Archived from the original on 3 July 2007. http://en.wikipedia.org/wiki/.NET_Framework. Retrieved May 2009.). The object is considered to be in use by the CLR as long as there exists a reference to that object, which might be either a direct reference to an object or via a graph of objects. When any reference indicates to an object, and it cannot be reached or used, it becomes garbage. However, it still keeps the memory allocated to it. That is why .NET Framework includes a garbage collector which runs periodically, on a separate thread from the application's thread, that finds all the garbage objects and reclaims the memory allocated to them.

The .NET Garbage Collector (GC) is a non-deterministic, compacting, mark-and-sweep garbage collector. (http://en.wikipedia.org/wiki/.NET_Framework, Referred to on May, 2009) The GC runs only when a certain amount of memory has been used or there is a large demand for memory on the system. Since it is not guaranteed when the conditions to start GC arise, the GC runs are non-deterministic. Each .NET application has a set of pointers to objects on the managed heap (*managed objects*). These include

references to static objects and objects defined as local variables or method parameters, as well as objects referred to by CPU registers. When the GC runs, it pauses the application, and for each object referred to in the pointer, it recursively marks all objects, which are reachable from the root objects, as reachable. It uses .NET metadata and reflection to discover the objects encapsulated by an object. It then enumerates all the objects on the heap (which were initially allocated contiguously) using reflection. All objects not marked as reachable are garbage. This is the *mark* phase. Since the memory held by garbage is not of any consequence, it is considered free space. However, this leaves chunks of free space between objects which were initially contiguous. The objects are then *compacted* together, by using `memcpy` to copy them over to the free space to make them contiguous again. Any reference to an object invalidated by moving the object is updated to reflect the new location by the GC. The application is resumed after the garbage collection is over. (ibid.)

The GC used by .NET Framework is actually *generational*. ("Garbage Collection—Part 2: Automatic Memory Management in the Microsoft .NET Framework". Archived from the original on 26 June 2007. http://en.wikipedia.org/wiki/.NET_Framework. Retrieved May 2009.) That means that objects are assigned a *generation*; newly created objects belong to *Generation 0*. The objects that survive a garbage collection are moved to *Generation 1*, and the Generation 1 objects that survive another collection are becoming *Generation 2* objects. Objects from higher generation are garbage collected less frequently than objects from lower generations. This mechanism increases the efficiency of garbage collection because older objects tend to have a larger lifetime than newer objects. Thus, by removing older (and thus more likely to survive a collection) objects from the scope of a collection run, fewer objects need to be checked and compacted (ibid.).

In the developing process .NET Framework version 3.5 Service Pack 1 was used which provides the following new features and improvements as written in Overview to Microsoft .NET Framework 3.5 Service Pack 1:

- “ASP.NET Dynamic Data, which provides a rich scaffolding framework that enables rapid data driven development without writing code, and a new addition to ASP.NET AJAX that provides support for managing browser history (back button support). For more information.

- Core improvements to the CLR (common language runtime) that include better layout of .NET Framework native images, opting out of strong-name verification for fully trusted assemblies, improved application startup performance, better generated code that improves end-to-end application execution time, and opting managed code to run in ASLR (Address Space Layout Randomization) mode if supported by the operating system. Additionally, managed applications that are opened from network shares have the same behavior as native applications by running with full trust.
- Performance improvements to WPF (Windows Presentation Foundation), including a faster startup time and improved performance for Bitmap effects. Additional functionality for WPF includes better support for line of business applications, native splash screen support, DirectX pixel shader support, and the new WebBrowser control.
- ClickOnce application publishers can decide to opt out of signing and hashing as appropriate for their scenarios, developers can programmatically install ClickOnce applications that display a customized branding, and ClickOnce error dialog boxes support links to application-specific support sites on the Web.
- The Entity Framework is an evolution of the existing suite of ADO.NET data access technologies. The Entity Framework enables developers to program against relational databases in according to application-specific domain models instead of the underlying database models. The Entity Framework introduces some additional features, including support for new SQL Server 2008 types, default graph serialization of Entities, and the Entity Data Source. This release of the Entity Framework supports the new date and file stream capabilities in SQL Server 2008. The graph serialization work helps developers who want to build Windows Communication Foundation (WCF) services that model full graphs as data contracts. The Entity Data Source provides a traditional data source experience for ASP.NET Web application builders who want to work with the Entity Framework.
- LINQ to SQL includes new support for the new date and file stream capabilities in SQL Server 2008.
- The ADO.NET Data Services Framework consists of a combination of patterns and libraries, which enable data to be exposed as a flexible REST (Representational State Transfer)-based data service that can be consumed by

Web clients in a corporate network or across the Internet. The ADO.NET Data Services Framework makes data service creation over any data source. A conceptual view model of the underlying storage schema can easily be exposed through rich integration with the ADO.NET Entity Framework. Services created by using the ADO.NET Data Services Framework, and also compatible Windows Live (dev.live.com) services, can be easily accessed from any platform. For client applications that are running on Microsoft platforms, a set of client libraries are provided to make interaction with data services simple. For example, .NET Framework-based clients can use LINQ to query data services and a simple .NET Framework object layer to update data in the service.

- Windows Communication Foundation now makes the DataContract Serializer easier to use by providing improved interoperability support, enhancing the debugging experience in partial trust scenarios, and extending syndication protocol support for wider usage in Web 2.0 applications.
- The .NET Framework Data Provider for SQL Server (SqlClient) adds new support for file stream and sparse column capabilities in SQL Server 2008. “

(http://www.microsoft.com/downloads/details.aspx?FamilyID=AB99342F-5D1A-413D-8319-81DA479AB0D7&displaylang=en, Retrieved in September, 2009)

4.1.2 ADO.NET Entity Framework

ADO.NET Entity Framework is an object-relational mapping (ORM) framework for the .NET Framework which abstracts the relational schema of the data that is stored in a database and presents its conceptual schema to the application. For example, in the database, entries about a customer and their information can be stored in the Customers table, their orders in the Orders table and their contact information in yet another Contacts table on the other hand Customers, Orders and Contacts tables are represented by classes in an application. For an application to deal with this database, it has to know which information is in which table, that is why the relational schema of the data is hardcoded into the application.

The disadvantage of this approach is that if schema of the database is changed, the application also need the change. “Also, the application has to perform SQL joins to traverse the relationships of the data elements in order to find related data. For example, to find the orders of a certain customer, the customer needs to be selected from the Customers table, joined with the Orders table, and then projected to remove unwanted columns.” (http://en.wikipedia.org/wiki/Entity_Framework, Referred to on June, 2009)

Object-oriented programming languages, where the relationships of an object's features are given to the user as Properties of the object and accessing the property traverses the relationship is completely different from the model of traversing relationships between items. Furthermore, using SQL queries expressed as strings, does not give guarantees about the operation and does not provide compile time type information which can easily lead to exceptions during the execution of the application.

In this model the client side data access mechanisms are shielded from mapping of the logical schema into the physical schema that defines how the data is structured and stored on the disk because it is the job of the database system as the database exposes the data in the way specified by its logical schema.
(http://en.wikipedia.org/wiki/Entity_Framework, Referred to on June, 2009)

4.1.3 LINQ

LINQ stands for Language Integrated Query. It is a set of extensions to the .NET Framework that encompass language-integrated query, set, and transform operations. It extends C# with native language syntax for queries and provides class libraries to take advantage of these capabilities. It introduces standard, easily-learned patterns for querying and updating data, and the technology can be extended to support potentially any kind of data store. Visual Studio 2008 includes LINQ provider assemblies that enable the use of LINQ with .NET Framework collections, SQL Server databases, ADO.NET Datasets, and XML documents.. Here only LINQ to Objects will be

presented. The main idea of LINQ is to allow developers to focus more on functionality rather than on creating repetitive code. LINQ syntax is very similar to SQL and it has a lot of the same keywords and similar functionalities. (Nash, 2007, 465-467).

The following listing shows basic LINQ query:

```
string[] names = { "Burke", "Connor", "Frank",
                   "Everett", "Albert", "George",
                   "Harris", "David" };

IEnumerable<string> query = from s in names
                            where s.Length == 5
                            orderby s
                            select s;

foreach (string item in query)
      Console.WriteLine(item);
```

(101 LINQ samples, Referred to in September, 2009, http://msdn.microsoft.com/en-us/vcsharp/aa336746.aspx)

The query is searching for strings with length 5, in the array. It starts from the declaration (the order of variables is random) of table of strings (names); later there is a condition and final result value. In comparison with an SQL query there would be "select" statement at the beginning, next the table name and the condition at the end. In LINQ the order is opposite, though the IntelliSense can still work when the query is constructed (the type is well known because it is retrieved from the type of collection elements). The query returns `IEnumerable<string>`. LINQ queries can be changed into lambda expressions tree which is implementation of anonymous methods. From version 2.0 of .NET Framework Lambda expression are occurring in C# language as more compact way of using anonymous methods.

```
IEnumerable<string> query = names
                            .Where(s => s.Length == 5)
                            .Select(s => s.ToUpper());
```

In method Where in brackets before '=>' sign, there are parameters that should be used in expression and after sign the operation which should be performed on parameters.

This syntax can be used without any loss in readability if the query is simple, but with more complex queries, more readability have standard LINQ query syntax .

(Kumar, LINQ Quickly, 2007, Referred to on October, 2009)

4.1.4 ASP.NET

ASP.NET is a web application framework developed and marketed by Microsoft to allow programmers to build dynamic web sites, web applications and web services. It was first released in January 2002 with version 1.0 of the .NET Framework, and is the successor to Microsoft's Active Server Pages (ASP) technology. ASP.NET is built on the Common Language Runtime (CLR), allowing programmers to write ASP.NET code using any supported .NET language. (http://en.wikipedia.org/wiki/Asp.net, Retrieved on September, 2009)

ASP.NET includes applications, extensions to applications and a big part of the .NET class hierarchy. ASP.NET works under Microsoft IIS web server. In HyperNotifier solution it will be using Microsoft Windows 2003 Server and IIS to host ASP.NET application.

.NET pages, known officially as "web forms", are the main building block for application development. (MacDonald and Szpuszta, 2007 p. 63) Web forms are consist of aspx files, these files typically contain static (X)HTML language, as well as Web Controls and User Controls where the required content for the web page is added by the developers. Additionally, in similarity to other web development technologies such as PHP, JSP, and ASP; dynamic code which runs on the server can be placed in a page within a block `<% -- dynamic code -- %>`, but this practice is generally not used because of data binding since it requires more calls when rendering the page.

However dynamic code can be placed in a page, Microsoft recommends using the code-behind model for dealing with dynamic program code, which places this code in a separate file or in a specially designated script tag. Code-behind files are named like *MyPage.aspx.cs* or *MyPage.aspx.vb,* so the first part of the filename is the same as aspx file including extension, and second part is an extension denoting the page

language. In Microsoft Visual Studio and other IDEs this practice is automatic. This style of programming allows the developer to write code that responds to different events, like e.g. the page loading, or a click on a control; rather than a step by step walk through the document.

ASP.NET's code-behind model is a departure from Classic ASP because it leads programmers to build applications which separate presentation layer from content. In theory, this would allow a web developer to focus on the design part, which gives less opportunity for distortion of the program code that is underneath. This is similar to the separation of the view from the controller in MVC (Model-View-Controller) design pattern.

The rendering methods used in ASP.NET are called *visited composites*. The Web form file (.aspx) is compiled into initialization code which builds the composite (control tree) representing the original template. Server controls are represented by instances of a specific control class. The code that initializes the page is a combination of user-written code and results in a class specific for the page. The written code is usually the assembly of multiple partial classes.

The requests for the page are processed through the following steps. The first is initialization, when an instance of the page class is created and the initialization code is executed. In the next steps the produced initial control tree is manipulated by the methods of the page. Because the node in the tree is a control represented as an instance of a class, the tree structure can be change by the code and manipulation of the properties/methods of the individual nodes can occur. Finally, the rendering step is executed, a visitor is asking each node visited in the tree to render itself using its methods. The result is HTML code, which is sent to the client.

For a programmers that are novice in ASP.NET who are used to rely on class instance members, the page lifecycle can be very confusing, because after the request processing has ended, the instance of the page class is discarded and with it the entire control tree, so all the data stored in controls is lost with every page request/response cycle.

(http://en.wikipedia.org/wiki/Asp.net, Retrieved on September, 2009)

4.2 Web Part

A **web part** is an ASP.NET server control which can be added by users at run time to a Web Part Zone on Web Part Pages. Web Parts are giving the user ability to modify the content, appearance, and behavior of Web pages directly from a browser. They are an integrated set of controls for creating Web sites. The User Interface controls derive from the **Part** class, and they compose the primary UI on a Web Parts page.

Web Parts are Microsoft's implementation of Web Widgets idea.

Web Parts can be used as add-on ASP.NET technology to Windows SharePoint Services. (http://en.wikipedia.org/wiki/Web_part, Retrieved on September, 2009)

Web Parts are equivalent to Portlets, but do not necessarily require a web portal such as SharePoint to host them.

(http://en.wikipedia.org/wiki/Web_part, Accessed on September, 2009)

5 CONCLUSION

5.1 Evaluation of the solution

The HyperNotifier project has ended successfully. All requirements have been meet and it is a usable and well constructed solution.

The time spent for developing HyperNotifier was long because it was a minor project in the company. There was only one developer working on this project, although in the designing part there were also three other people involved, the project manager, an analyst and a programming specialist who discussed the project during the implementation phase, sometimes adding some corrections. Some parts took more time for creating than expected, mostly because during the implementation better solution appeared or the requirements were changed. When smaller parts of the solution were complete, the team also discussed the current features, and if something was missing, it was added to the plan. There were also elements that were created faster than planned because of finding a better way to do it.

The solution consist of five project including five interfaces, one windows service, one web application and additionally three plugin projects. It is prepared to be as simple as possible for extension. It only needs to implement a small number of interfaces and build a DLL, then add it to the plugins folder in the project and add the name of the plugin into the configuration file. This is enough for the user to enjoy the new functionality. Also communication with the user interface has been stretched to the interfaces, which allows easy connection and a way to communicate with the user.

Although the HyperNotifier solution is doing what it should, and gives the user possibilities to accomplish the goals that were specified, there is still room for improvement, and new features. The next section explores possible further tasks which will give HyperNotifier new abilities.

Before HyperNotifier was built the changes were arrived but were not coordinated by anything. Every source had its own way to notify about changes, thus, different mails were arriving from many sources and it was hard to cope with all of them. Each mail had a different layout and sometimes arrived at a very surprising time. If mails needed

to be presented and formatted, someone had to gather them, organize, do the processing manually and prepare the report. The created solution is a big step toward uniformity and efficiency, an automated processing of information about the changes from all sources in the company.

The HyperNotifier is currently used to process incoming information, and scan source for changes if needed. The route of the information is automatic, and the arriving information ends up in the database and afterwards at specified time are sent by mail in a consistent and plain report. The users can then view this report and take appropriate action. They have easy overview through all the changes in all the sources and only those that they wanted to see.

Development will not stop, however, as there are now many new requirements present, which will be gradually introduced to the project.

The HyperNotifier project appeared even more complicated project than it was in the Assumption. There were a number of parts that could be made in at least two different ways like, for example, handling of the DLLs or database communication. The first of them was solved by choosing the easiest and least complicated solution which was using the System.Reflection.Assembly in opposite to System.AddIn library. The choice of database communication was not so simple. It was important to have an easy and especially fast access to the data and it was not desirable to use old solutions. In the end, the decision fell on Entity Framework with the help of LINQ mostly because it is quite new and sufficiently tested. In decisions like these there was also a significant influence of the desire to learn new technologies and it always was a big argument for.

5.1.1 Future plans

The Hypernotifier was designed in a way that making future changes should be an easy task, but still, of course, that depends on how complex the new functionality is. The solution is made to be extensible in plugins area so there can be added very easily many new sources of data.

The Hypernotifier already provides a way to work with the incoming changes and send outgoing reports. There are, however, many more aspects to working with the

changes, and this section can be explored for some of the possibilities that might be implemented in the future for the Hypernotifier.

The solution is designed to continuously expand and improve, thus after the release of the first version the improving work is started. The nearest plan is to add new plugins for a number of additional sources and to move the user interface to the web part to allow placing it on SharePoint site.

So features that can extend the functionality of Hypernotifier are as follows:

- Adding new plugins for new sources of data.
- New user interface (changeable) like Webpart.
- Adding some new functionality to the UI to make it more user friendly (e.g. fast searching).
- Adding new premade templates for reports.
- Giving the user more capabilities to design report (e.g. organizing data on the report and designing layout).

The very big step ahead from only notifying about changes could be adding to Hypernotifier the functionality to prepare reports from all the gathered data comparing, for example, which files are commonly changed and which very rare. That kind of information can be also very useful in a company like VSoft.

5.2 Personal experience

I am sure that I definitely have learned a lot during the development. I am also very proud of what I did and how it improved me as a professional. The Hypernotifier was my biggest and most complex project. After having finished it I know more about designing distributed applications, what kind of solutions should be used, which should be avoided. However, still I see the areas where I need to develop myself.

The Hypernotifier as a team project in some ways gave me also a lot of new experience. Now I understand how important good communication between project members is and how to make team work more efficient. Also I realized how hard is to make a good software which will be satisfactory for the customer. It requires

continuous consultation and comparing the vision of the developers with the vision of the future user. Working closely together with those who will be using the solution was in my opinion the best solution. They were constantly inspecting the work, the initial requirements laid out for the solution were dynamically changing, and the application needed to adapt to these new requirements.

Creating distributed applications differs in many fields from making simple, condensed programmes. The different parts need to be carefully integrated together, so they can work seamlessly with each other. Additionaly, the big challenge of today's programmers is to make the parts easily changeable. That is not as easy as it might first seem to create a software that can be easily modified and manipulated. The most important issue is careful planning and continuous communication with the customer, without it the changes added to the solution might overwhelm the creator, and not meet the customers' expectations.

6 REFERENCES

Weldon W. Nash, Accelerated C# 2008, Referred to on June, 2009.

Clare Churcher, Beginning Database Design, 2007, Referred to on April, 2009.

Paul Wilton and John W. Colby, Beginning SQL, 2005, Referred to on April, 2009.

Joseph Albahari, Ben Albahari, LINQ Pocket Reference, 2008, Referred to on October, 2009.

N Satheesh Kumar, LINQ Quickly, 2007, Referred to on October, 2009.

Matthew MacDonald and Mario Szpuszta, Pro ASP.NET 3.5 in C# 2008 Second Edition, 2007, Referred to on May, 2009.

Simon Robinson, Christian Nagel, Jay Glynn, Morgan Skinner, Karli Watson, Bill Evjen, Professional C# Third Edition, 2004, Referred to on May, 2009.

Robert Vieira, Professional SQL Server™ 2005 Programming, 2007, Referred to on April, 2009.

Charles Petzold, Programming Microsoft Windows with C#, 2002, Referred to on April, 2009.

101 LINQ samples, Referred to on September, 2009, http://msdn.microsoft.com/en-us/vcsharp/aa336746.aspx .

VSoft, Referred to on April, 2009, http://www.vsoft.pl/index.php?option=com_content&view=article&id=1&Itemid=13.

Windows service, Referred to on April, 2009, http://en.wikipedia.org/wiki/Windows_service.

Introduction to Windows Service Applications, Referred to on April, 2009, http://msdn.microsoft.com/en-us/library/d56de412%28VS.80%29.aspx.

.NET Framework, Referred to on May, 2009, http://en.wikipedia.org/wiki/.NET_Framework.

.NET Framework Technologies, Referred to on May, 2009,

http://msdn.microsoft.com/en-gb/netframework/default.aspx.

ADO.NET Entity Framework Overview, Referred to on June, 2009,

http://msdn.microsoft.com/en-us/magazine/cc163399.aspx.

ADO.NET Entity Framework, Referred to on June, 2009, http://en.wikipedia.org/wiki/Entity_Framework.

ASP.NET, Referred to on September, 2009, http://en.wikipedia.org/wiki/Asp.net

LINQ, Referred to on November, 2009, http://msdn.microsoft.com/en-us/netframework/aa904594.aspx.

Web part, Accessed on September, 2009, http://en.wikipedia.org/wiki/Web_part.

What's New in the .NET Framework Version 3.5, Referred to on September, 2009,

http://msdn.microsoft.com/en-gb/library/bb332048.aspx.

Garbage Collection: Automatic Memory Management in the Microsoft .NET Framework, Archived from the original on 3 July 2007, Retrieved on May, 2009, from http://en.wikipedia.org/wiki/.NET_Framework.

Garbage Collection—Part 2: Automatic Memory Management in the Microsoft .NET Framework, Archived from the original on 26 June 2007, Retrieved on May, 2009, from http://en.wikipedia.org/wiki/.NET_Framework.