Targeted News in an Intranet

LISAN CHEN
and
TINGTING SCHILLER SHI

KTH Information and Communication Technology

Degree project in
Communication Systems
Second level, 30.0 HEC
Stockholm, Sweden

Targeted News in an Intranet

Lisan Chen
Tingting Schiller Shi

2013-10-29

Master Thesis Report

Examiner and academic adviser
Professor Gerald Q. Maguire Jr.

School of Information and Communication Technology (ICT)
KTH Royal Institute of Technology
Stockholm, Sweden

Thesis project taking place at
How Solutions Stockholm AB
Stockholm, Sweden

Abstract

In SharePoint 2013, Microsoft added a social networking function in the personal sites (*My Site*) of a user. In this version, a personal news feed has been added which shows events regarding subjects the user follows, such as document changes, user updates, tagged posts, and site activities. The purpose of the thesis is to investigate whether or not it is possible to extend the news feed function by adding an independent component as part of *My Site*, to allow users to follow corporate news by choosing their categories of interests.

A prototype of the component was implemented and it met most of the objectives stated in the thesis. It is added to the default page of the user's *My Site* as a web part and it is able to retrieve and display news that matches the user's subscription. Although the web part still needs improvements in both functionality and design, it still confirms that it is possible to extend the current *My Site* news feed with such a component.

Since the students working on this thesis had no prior knowledge of SharePoint or .NET development, the project brought new challenges, as the students needed to learn how to work in a SharePoint environment and to learn to use Microsoft Visual Studio for .NET development.

Keywords: SharePoint 2013, news feed, independent component, corporate news, web part, My Site, Microsoft Visual Studio, .NET

Sammanfattning

Microsoft har i SharePoint 2013 förbättrat användarnas personliga sidor (My Site) genom att sammankoppla dem i ett socialt nätverk. I förbättringen har ett personligt nyhetsflöde som visar händelser som användaren följer tillagts. Denna rapport avser att undersöka möjligheten att utöka det personliga nyhetsflödet med att lägga till en oberoende komponent i My Site. Komponenten ska tillåta användarna att prenumerera på företagsnyheter genom att välja bland olika nyhetskategorier.

En prototyp av komponenten implementerades och resultatet uppfyllde de flesta kraven som ställdes i början av arbetet. Komponenten har lagts till i användarens My Site som en webb del och hämtar automatiskt de senaste företagsnyheterna som matchar användarens prenumeration. Den utvecklade prototypen kan förbättras både i funktion och design, men har uppfyllt behovet för denna rapport som avser att undersöka möjligheten att utöka det personliga nyhetsflödet i *My Site* med en sådan komponent.

Eftersom projektmedlemmarna saknade förkunskaper i SharePoint och .NET utveckling innebar projektet nya utmaningar. Studenterna lärde sig att arbeta i SharePoint miljö samt i Microsoft Visual Studio för .NET utveckling.

Nyckelord: SharePoint 2013, nyhetsflöde, oberoende komponent, företagsnyheter, webbdel, My Site, Microsoft Visual Studio, .NET

Acknowledgements

We would like to thank our examiner and academic adviser Professor Gerald Q. Maguire Jr. for the suggestions and feedback that we have received during this period of time. His advices have been very inspirational and helpful.

We would also like to thank Mattias Kjörk at HOW Solutions for this great work experience and the opportunity to challenge ourselves in a new field of work. Mostly, we would like to thank everyone at HOW Solutions for their support and suggestions when we had problems proceeding with the work.

Lastly, we would like to thank our families and friends for the never ending support and encouragements, which helped us through the most difficult times.

Table of contents

List of Figures

List of Acronyms and Abbreviations

AIIM	Association for Information and Image Management
API	Application Programming Interface
CA	Central Administration
CSS	Cascading Style Sheet
DLL	Dynamic-Link Library
IIS	Internet Information Services
IIS	Internet Information Services
ISAPI	Internet Server Application Programming Interface
MMS	Managed Metadata Service
OATH	Open Authentication
RSS	Rich Site Summary
SA	Service Application
SSP	Shared Services Provider
WA	Web Application
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language

1 Introduction

This chapter gives a short introduction to the area, as well as a longer definition of the problem addressed in this thesis project. This is followed by a statement of the goals to be achieved within this thesis project and a description of the general structure of the thesis.

1.1 General Introduction to the Area

SharePoint is a widely used multipurpose web application platform developed by Microsoft. It specifically targets enterprises and over a third of all organizations of the 674 survey responses by Association for Information and Image Management (AIIM) members in 2011 [1] use SharePoint for content management across the enterprise. Initially, SharePoint mainly focused on intranet content and document management. However, the most recent versions of SharePoint have more wide-ranging capabilities [2], where in addition to intranet portals and document & file management, SharePoint can also provide organizations with social networks, extranets, collaborative services, websites, enterprise search, and business intelligence services.

In SharePoint 2010 and SharePoint 2013, users have personal sites called “My Site”. Compared with the earlier versions, SharePoint 2013 has improved the users’ My Sites by interconnecting them in a social network. Microsoft added various capabilities for social networking [3] in SharePoint, such as news feeds, SkyDrive Pro, community sites, and task list aggregation. A user can also choose to follow content or people of interest on the intranet.

The news feed in My Site shows a list of the latest events. These events consists of content the user has chosen to follow. As mentioned above, the user can choose to follow specific people and/or content, such as documents, tags, or sites which are available in the intranet. When a change occurs in a document or when a new conversation starts, the event will be shown in the user’s news feed. Figure 1 illustrates an activity in the news feed of My Site using SharePoint 2013.

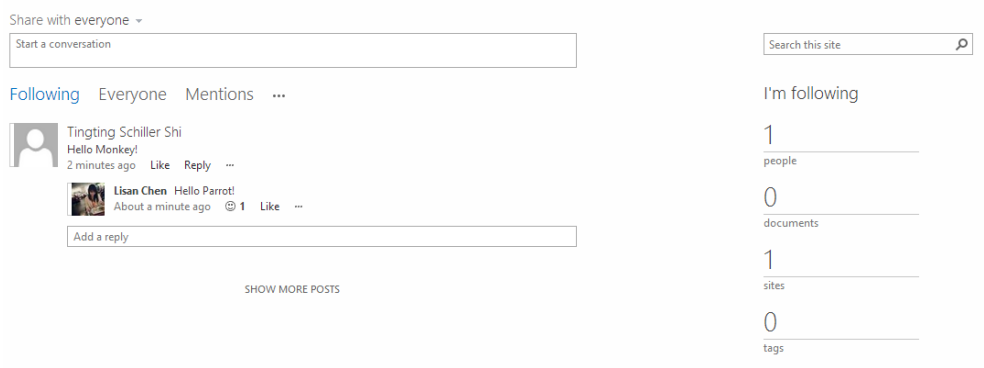

Figure 1. A conversation event shown as a newsfeed in My Site

1.2 Problem Definition

The purpose of this thesis project is to investigate whether or not it is possible to extend the news feed function in a SharePoint 2013 environment by adding an optional news component to enable users to subscribe to specific corporate news categories such as IT, business, or economy.

The component should also allow the company to add relevant metadata to each news article, such as its news category and the name of a specific department. If the news article is targeted towards all employees, then it should be shown in all users' feeds regardless of their subscriptions.

1.3 Goals

The generic goals of this thesis project are:

- Design and implement a prototype of the proposed targeted news component.
- Demonstrate that the component shows news *relevant* to the user.

The specific goals of this thesis project are:

- The component should be added as an independent part of My Site.
- A user should be able to subscribe to their selected news categories.
- The company should be able to add news articles to the different categories.

- If a news article is relevant to the user, for example if the article targets the user's department, then the article should be visible via the news component.

1.4 Structure of the Thesis

Chapter 2 gives some background about existing solutions and a brief review of general knowledge about the SharePoint architecture as needed to fully understand this thesis. Chapter 3 describes the SharePoint foundation and the existing services it was build upon. The method used in this thesis, as well as the results and analysis is presented in Chapter 4 and Chapter 5 respectively. Chapter 6 shows the conclusion of this thesis followed by suggestions for future work. A list of references can be found at the end of the thesis.

2 Background

Along with the growth of news-related websites, more and more people use tools in order to gather their news subscriptions in one place. This way, users save time as they do not need to individually visit these separate websites. Some existing solutions related to this project's topic will be briefly presented in section 2.1. An overview of the SharePoint architecture is presented in section 2.2 and the prerequisites to understand this thesis is given in section 2.3.

2.1 Earlier work

This section will present a selection of existing aggregation techniques that are related to the topic of this project.

2.1.1 RSS and Atom feed readers

News feed readers are popular tools to subscribe to news via the Internet. The number of news publishers who syndicate their site contents as Rich Site Summary (RSS) or Atom feeds is growing rapidly [4]. RSS and Atom are two different XML formats that are used for web feeds. Web feeds allow software feed readers to receive web contents [5]. It does not matter which format the publishers choose when publishing web contents, since both formats serve the same function. The two formats are both simply special types of web pages, which users subscribe to. Generally, news stories are grouped by category (Business, Sports, etc.), where each category is distributed as a different RSS or Atom feed. Using feed readers, users can subscribe to news feeds in order to receive updates concerning their selected category or categories of interest [6].

Sites offering RSS or Atom feeds include Google News, Yahoo News, and CNN. Various native feed readers are available for different platforms, such as Amphetadesk (Microsoft Windows, Linux, Apple's Mac OS), FeedReader (Windows), and NewsGator (Windows Outlook). Other popular web-based feed readers are My Yahoo, Bloglines [4], and NewsGator Online Services [10]. Figure 2 shows an example of a web-based feed reader, in this case Google Reader (Note: Google Reader was discontinued 1st July 2013 [20]).

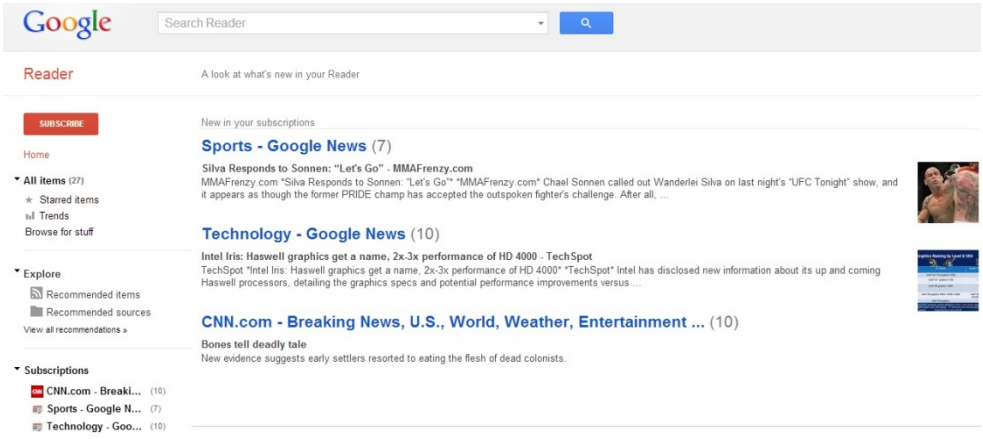

Figure 2. Google Reader

2.1.2 News Rollup Web Part

The purpose of using RSS and Atom feeds is similar to the purpose of this project; to allow users to easily subscribe to and receive news updates. Both types of feeds are used by a wide range of websites. Since this thesis concerns developing a SharePoint news component, this section will give a brief introduction of an existing SharePoint solution called News Rollup Web Part.

In SharePoint, news can be published as announcements using announcements lists. An announcements list is created by default when creating a SharePoint site. The list appears as a web part (view) on the user's home page and it typically displays the five most recently published announcements. Older announcements disappear from the web part, but they are still accessible via the All Item view of the Announcements list [7]. Typically, web applications have at least one site collection, which is a collection of SharePoint sites [8]. Users that have access to more than one site in a site collection may want to follow several announcements lists, one for each site. However, it can be time consuming and inefficient to view each site individually. The News Rollup Web Part solves this problem by displaying the most recent announcements of each site within the current site collection. This approach provides better visibility of new announcements from the announcement lists of the sites. Customization of content layout is provided to allow users to make their own modifications, such as defining the number of words and number of announcements to be displayed, customizing layout using CSS, and showing or hiding the author's picture. As illustrated in Figure 3, the web part displays some small amount of information about the announcements with links to the actual announcement pages [9].

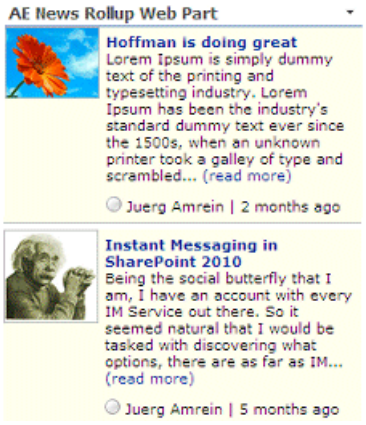

Figure 3. News Rollup Web Part view (Appears here with the permission of Amrein Engineering. The figure originally appeared as figure AE News Rollup Web Part on [21].)

2.1.3 RSS Viewer Web Part

As mentioned in section 2.1.1, using a RSS feed is a popular method among news publishers when publishing information on the Internet. Content in SharePoint (such as libraries, lists, and documents) can also be syndicated as an RSS feed, thus a SharePoint user can subscribe to feeds and get updates automatically using a feed reader. However, some SharePoint users may want to view the feeds on a SharePoint site such as My Site. Instead of using one of the regular feed readers mentioned in section 2.1.1, users can add a RSS Viewer Web Part as a part of My Site. By adding this web part, a user can view all subscribed RSS feeds directly on My Site. The web part can display both external subscriptions (such as sports news and weather reports) and content updates within the SharePoint site collection. RSS Viewer offers convenience for those who prefer to view all of the information from different sources via a single SharePoint page [11].

2.1.4 Virto Social Aggregator Web Part

Similar to RSS Viewer, Virto Social Aggregator is a SharePoint web part that combines RSS, Atom, blogs, and tweets into a single view. This component is compatible with SharePoint 2007 and SharePoint 2010 and it provides full user interface customization using XSL and CSS. Users can customize the overall layout, feed item layouts, and stylesheet. In addition to aggregation of feeds, the component offers additional features for Twitter users allowing them to both read and post tweets right from SharePoint. Open Authentication (OATH) protocol is supported by Virto Social Aggregator and this method of authentication is accepted by Twitter, thus the web part does not store the users' accounts or passwords. SharePoint administrators have the ability to give users access to these Twitter features or prevent them from using these features [12].

In the future, the developers of Virto Social Aggregator plan to add integration with Facebook in order to allow users to get and post information from their Facebook network. Another feature that will be added is the support for approval workflow, which gives the possibility to define an approval process for all posts [12].

2.1.5 Content Query Web Part

As presented in section 2.1.2, the News Rollup Web Part aggregates announcements from several announcements lists within a site collection. However, the aggregator does not filter the results to adapt to the users' criteria, nor does it access information from other lists and libraries; therefore the web part may not satisfy users who want to subscribe to more specific information than is available from these announcements lists.

Intranets based on SharePoint typically use document libraries and lists to share information among users. For example, a project group can create a library or a list of common documents for the project's team site. Only team members and those that are given permission to access the site can access the documents. Libraries and lists typically come with the site. Users can either use the existing libraries and lists or to create new ones to share information [13]. What if a user wants to subscribe to multiple lists and libraries? And what if the user only wants to see the documents that were modified by a specific user? The Content Query Web Part is a solution for SharePoint which allows users to subscribe to documents in libraries and lists throughout a site collection. Users can query all documents in a site collection, in one site and all of its sub-sites, as well as in a single list or library. Filters can be added to the queries to match the user's criteria. The user can also modify the results in several ways, such as grouping, sorting, and limiting the number of results to be displayed. This web part shows the most recent updated information that the user is authorized to see. The queries are run whenever the browser refreshes, which in turn automatically refreshes the query results [14].

2.1.6 Proactive News Module

Proactive provides a news module that targets news on SharePoint intranets. The module targets relevant news contents to the relevant users based on their properties such as divisions, departments, teams, and individuals. This module gives users a personalized view of news when they log onto the intranet [22].

The Proactive News Module provides a user-friendly tool to publish news from many different news channels. Users are divided into groups that subscribe to relevant news. A user can also decide which news channel to subscribe to individually. The overall functions of the module are listed below [22]:

- Target relevant news to relevant group of audiences, individual business units, and individuals.
- Publish news to different levels of the organization.
- Publish many different content types, such as text, images, videos, etc.
- Users can comment on and rate news.
- Users can share news links with others.
- Users can subscribe to news and updates.

2.2 Prerequisites

This thesis is targeted to anyone interested in developing independent components in SharePoint 2013. This thesis will only cover a small part of the SharePoint environment. Readers of this thesis do not need any prior knowledge of the SharePoint architecture. All of the relevant information regarding SharePoint will be presented in this thesis.

3 Microsoft SharePoint Architecture & Topology

SharePoint is a very flexible platform that offers scalability, as it can be run on a single machine or across hundreds of machines [15]. This chapter aims to describe the foundation of Microsoft SharePoint and its logical architecture and topology.

3.1 Microsoft SharePoint Foundation

This chapter will describe in detail how the foundation for Microsoft SharePoint is composed. A description of Microsoft's Internet Information Services (IIS) and the ASP.NET Framework will be given in section 3.1.1 and 3.1.2 (respectively). Section 3.1.3 explains how the Windows SharePoint Services are integrated with these other two components, thus creating the foundation for Microsoft SharePoint.

3.1.1 IIS Web Sites and Virtual Directories

To understand the *Microsoft SharePoint* architecture, it is first necessary to understand the basic concepts behind an IIS Web site and virtual directories. Both *Microsoft SharePoint* and *ASP.NET* depend upon IIS 6.0 web server to handle incoming HTTP requests. In addition to handling incoming HTTP requests, IIS also provides a management infrastructure to start and run processes on the web server.

Each IIS Web site acts as an entry point into the IIS web server's infrastructure. IIS Web sites are configured to listen for and handle incoming HTTP requests that meet certain criteria. For example, an IIS Web site can be configured to handle requests coming over a certain IP address or port number.

A default IIS Web site, *Default Web Site* is automatically created and the IIS 6.0 web server is configured to listen for requests coming over TCP port 80 for all IP addresses supported by the web server. In addition to the *Default Web Site*, other IIS Web sites can be created using the IIS administration tools. As for any other IIS Web site, the *Default Web Site* defines a specific URL space that follow the pattern: http://www.Litwareinc.com/*[1]. An endless number of URLs can be created

[1] This URL is for a fictional company named Litware Inc. and is the name used throughout the Microsoft documentation for IIS.

within this URL space, and IIS handles incoming requests for these URLs by routing them towards the *Default Web Site*.

Every IIS Web site maps to a physical root directory within the web server's file system. IIS by default maps *Default Web Site* to the root directory in *C:\Inetpub\wwwroot*. The incoming HTTP requests can reference physical files in the root catalog defined by IIS. For example, when a request comes for the page http://www.Litwareinc.com/page1.htm/, IIS will respond by simply loading the content from the file *C:\Inetpub\wwwroot\page1.htm* into memory and sends this content to the client.

An important part of an IIS Web site is the ability to control whether incoming requests require authentication, and which authentication protocols should be used. For instance, a company can separate their internal network from the external network by simply changing the configuration for the IIS Web sites to be used for the internal network from the configuration used for the external network. A company might use the *Default Web Site* as its public Web site, i.e., as a website that can be accessed by everyone. In this case, the IIS Web site is configured to allow anonymous access and to support basic authentication. Other IIS Web sites can be created for internal use in the company, and configured to forbid anonymous access. In this case basic authentication is then replaced by Integrated Windows authentication [23].

Beyond the possibilities to create IIS Web sites, IIS also supports creation and configuration of virtual directories. A virtual directory is a logical entity that defines a child URL space nested inside the parent URL space. As with any IIS Web site, the virtual directory is also mapped to a physical directory on the web server. What makes a virtual directory different from a regular directory is that IIS provides greater flexibility to define the location for the root directory of a virtual directory. For example, a virtual directory within the *Default Web Site* with the URL space http://www.Litwareinc.com/Products/ could be configured to have its root directory in *C:\WebApps\Site1* instead of the *default C:\Inetpub\wwwroot\Products*.

IIS tracks all changes made in the IIS Web sites and virtual directories. These changes are saved as entries in the IIS metabase located in the file system of every front-end web server running IIS [23].

3.1.1.1 ISAPI Extensions and ISAPI Filters

In the most straightforward routing scenarios, incoming requests are mapped by IIS to a physical file in the root directory for the IIS Web site or to a virtual directory. IIS supports the *Internet Server Application Programming Interface* (ISAPI) programming model, which allows for more sophisticated request routing scenarios. The ISAPI programming model provides the possibility to configure an IIS Web site or virtual catalog to trigger the execution of custom code on the web server with incoming requests.

The original version of IIS introduced the ISAPI programming model. This application programming interface (API) still offers the lowest level for development of custom components for IIS. The ISAPI programming model consists of two key component types: *ISAPI extensions* and *ISAPI filters*.

An *ISAPI extension* is a component realized as *Dynamic-Link Library* (DLL) that acts as an endpoint for an incoming request. The basic idea is that IIS can map incoming requests to a set of endpoints that will trigger the execution of code within the ISAPI extension DLL. The ISAPI extension DLL needs to be installed on the web server and configured as an IIS Web site or virtual directory. Configuration generally includes defining an association between specific file extensions and ISAPI extensions. This is done with the help of an IIS application map.

While the ISAPI extension acts as an endpoint, an ISAPI filter acts as an interceptor. An ISAPI filter is installed and configured as an IIS Web site. When an ISAPI filter is installed, it intercepts and processes all incoming requests that target that specific IIS website. The basic task of ISAPI filters is to process incoming requests before and after they are passed to the rest of the IIS Web site. ISAPI filters are typically created to provide low level functionality in an IIS Web site, for instance to provide custom authentication and request logging.

An example scenario of how the ISAPI extensions and ISAPI filters interact is depicted in figure 4.

Figure 4. The relationship between ISAPI extensions and ISAPI filters

However, custom development of ISAPI components is not very popular these days for several reasons. ISAPI components are hard to design, develop, and debug since they need to be written in unmanaged C++ and require complicated coding techniques for thread synchronization (amongst other things). Most developers prefer to work on a level above the ISAPI, where frameworks such as ASP and ASP.NET are available [23].

3.1.1.2 Application Pools and the IIS Worker Process

IIS offers a flexible infrastructure for management of the actual web request processing using worker processes by utilizing *application pools*. An *application pool* is a configurable unit that gives control over how IIS maps the IIS Web sites and virtual directories to instances of the *IIS worker process*. Instances of the worker process are launched with an executable named *w3wp.exe*. This name in the following chapter refers to a worker process.

The routing architecture of IIS is controlled through a device driver in the kernel-level of the operating system named *http.sys*. This device driver listens for incoming HTTP requests and using the information provided by the IIS metabase routes requests to the correct instance of *w3wp.exe*. This instance is part of the target application pool. When the *http.sys* determines that the target application pool is missing an active instance of *w3wp.exe*, it dynamically launches a new instance to process the request. Figure 5 shows

the relationship between the kernel-level device driver *http.sys* and the worker processes *w3wp.exe*.

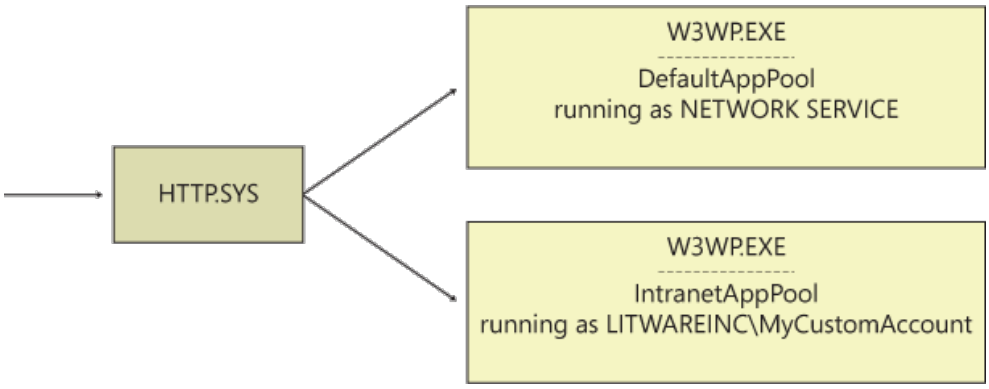

Figure 5. The relationship between http.sys and w3wp.exe (Used with permission from Microsoft. The figure originally appeared as figure 2-2 in [53].)

Every IIS Web site and virtual directory can be configured to run in its own isolated application pool. Conversely, it is also possible to configure multiple IIS Web sites and virtual directories to run in the same application pool for greater efficiency. An important aspect to consider is the tradeoff that exists between isolation and efficiency. Running multiple instances of the worker process gives greater isolation between the applications, but reduces the efficiency. Conversely, higher efficiency can be achieved by mapping multiple IIS Web sites and virtual directories to fewer instances of *w3wp.exe*, which in turn compromises their isolation.

Each application pool has an important setting known as the *application pool identity*. The *application pool identity* is configured with a specific Windows user account that is either a local account on the web server or a domain account in an *Active Directory* directory service domain. When *http.sys* starts a new instance of the *w3wp.exe* for a specific application pool, it uses the *application pool identity* for initialization of a Windows security token, which in turn is used as a process token. This setting is important because it establishes the "run as" identity for code that is executed in the worker process. As a result the code that executes in the worker process executes as if it were being run by this specific account. It is this binding between the account and the application pool that provides the isolation when two different accounts are used for two different application pools.

By default IIS uses the identity of the local Network Service account when an application pool is launched. However, it is possible to configure the *application pool identity* to use any account. When Web sites based on ASP.NET and Windows SharePoint Services are deployed it is recommended to configure the *application pool identity* with a domain account rather than a Network Service account. This is especially true in the

case of a Web farm environment when the identity of an application pool needs to be synchronized across multiple front-end web servers in the farm [23].

3.1.2 ASP.NET 2.0 Framework

The *ASP.NET Framework* provide a new layer of functionality on top of the IIS and ISAPI programming model. This framework provides the convenience and possibility to develop applications in a managed language, such as *Microsoft Visual C#* or *Visual Basic*. Additionally, the ASP.NET Framework provides the developer with valuable and helpful abstractions, for example data binding, navigation, state management and data caching.

The ASP.NET Framework is implemented as an ISAPI extension named *aspnet_isapi.dll*. As described previously, an ISAPI extension acts as en endpoint for incoming requests and it associates file extensions with specific ISAPI extensions by using application maps. The basic configuration for ASP.NET involves registration of application maps for common ASP.NET file extensions such as *.aspx*, *.ascx*, *.ashx,* and *.asmx*. This configuration is made on the same level as an IIS Web site or virtual directory. When IIS sees an incoming request with one of these extensions, the request is forwarded to the *aspnet_isapi.dll*, which passes control to the ASP.NET Framework. How the ASP.NET Framework processes the requests depends greatly on which extension the target has. The relationship between the worker process and the ASP.NET DLL is shown in figure 6.

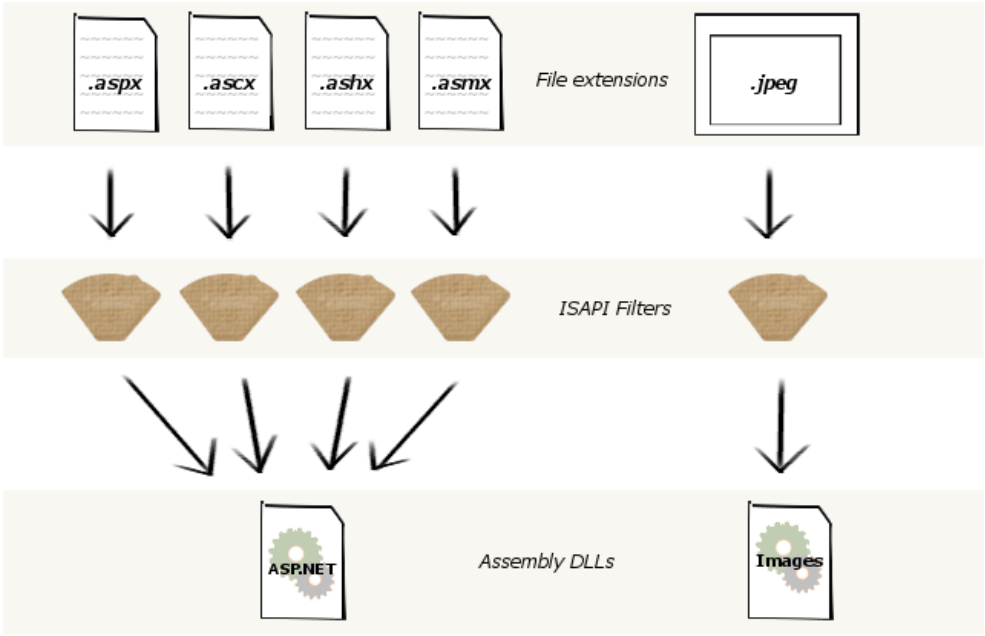

Figure 6. The relationship between ISAPI and ASP.NET

The ASP.NET Framework executes code to process each request for each IIS Web site and each virtual directory as an individual ASP.NET application. An ASP.NET application is logically a root directory for a set of files behind the application. This architecture promotes a very simple x-copy [24] style of deployment of ASP.NET applications. Creating a new virtual directory on the web server computer and copying the ASP.NET application files to the specified root directory is all that is necessary to deploy an ASP.NET application. However a lot more tedious work is required in a Web farm environment, since the virtual directory creation and file copying must be repeated on every front-end web server in the farm that is to provide this ASP.NET application.

Each ASP.NET application can be individually configured by adding a *web.config* file to the root directory. The *web.config* file is written in XML and specifies the configuration of the elements that control the behavior of several features in the ASP.NET Framework, for example compilation, state management, and page rendering.

The ASP.NET Framework runs each ASP.NET application with a certain level of isolation. This even applies to a scenario where multiple ASP.NET applications have been configured to run on the same IIS application pool. The ASP.NET Framework provides isolation between

ASP.NET applications that run on the same instance of *w3wp.exe* by loading each application into a separate .NET Framework *AppDomain* [24].

3.1.2.1 ASP.NET Pages

The ASP.NET page is one of the most appreciated concepts in the ASP.NET Framework. *Microsoft's Visual Studio* integrated development environment provides the possibility to visually construct pages for ASP.NET applications. Developers drag and drop server controls onto the visual design surface in Visual Studio, and modify the properties of pages and controls by utilizing standard property sheets. Additionally, the ASP.NET Framework and Visual Studio makes it moderately easy to add programming logic to pages by writing managed code, which executes in response to events on the control-level and page-level.

Fundamentally, an ASP.NET application page is realized as an *.aspx* file on the web server that is compiled into a DLL on request by the ASP.NET runtime. The content of an *.aspx* file may not be very complex, but compilation from an *.aspx* file to a DLL requires quite a bit of work (as will be described below).

First, the *.aspx* file contains definitions of all of the server-side controls and event handlers needed in the ASP.NET application. The *.aspx* file is parsed to generate a *Visual C#* or *Visual Basic* source file. This *Visual C#* or *Visual Basic* file contains a public class that inherits from the Page class defined within the *System.Web.UI* namespace. This namespace is defined inside the *system.web.dll* assembly. When the *ASP.NET page parser* generates this Page-derived class, a control tree of the defined server-side controls is built. The parser also adds the required code for hooking up the event handlers.

The ASP.NET page parser builds a source file for the *.aspx* page, which is then compiled into a DLL. The compilation happens automatically the first time a request comes in for this *.aspx* page. Once the DLL has been compiled by the ASP.NET runtime, this DLL can be used for all subsequent requests targeting that specific *.aspx* page. The ASP.NET runtime checks the date and time stamp on the *.aspx* file and retriggers the compilation process to rebuild the DLL when an updated version of the source file is found.

One reason for the immense popularity of the ASP.NET Framework is the convenience of server-side controls the framework offers. Pages can easily be composed with the help of out-of-the-box controls included in the ASP.NET Framework, such as *calendar control*, *validation controls,* and *data binding controls.* Another reason developers enjoys working with the ASP.NET Framework is because it is relatively simple to develop custom controls to use on ASP.NET pages.

When ASP.NET 2.0 was released in November 2005 [25], it introduced the concept of *master pages*, a very effective approach to page templating.

A master page is used across many different pages and it defines the common elements used in these pages, such as the top banner as well as site navigation controls. Every page linked to the master page makes use of the layout designed in the master page. A page linked to the master page is generally known as a *content page* in ASP.NET terminology. Figure 7 shows the relationship between a master page and its content pages.

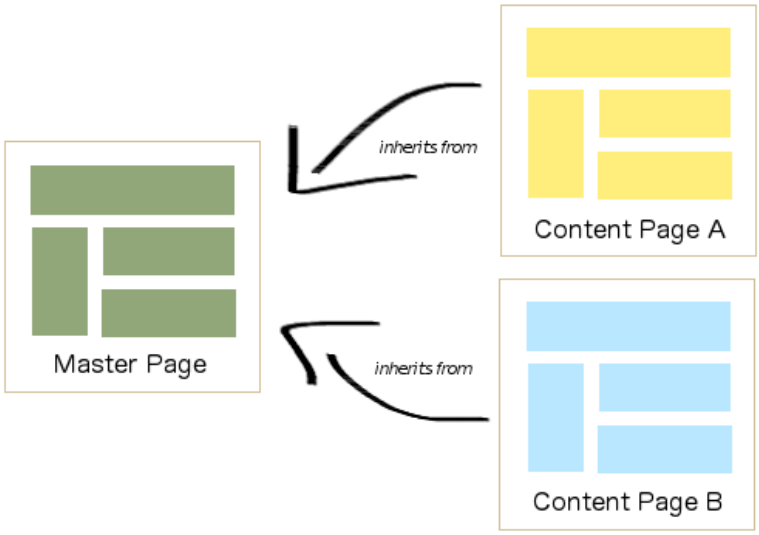

Figure 7. The relationship between a master page and content pages

For example, assume that you create a master page with the HTML layout shown in figure 7. The HTML layout consists of a top banner, a left side navigation bar and two content placeholders. Next you create a site collection named Colors with two content pages that utilize this master page. The two content pages contain blue and yellow placeholder (as shown in figure 7).

Figure 8 shows content page A where the top banner, left side navigation, and Placeholder B are colored yellow while Placeholder A is blue.

Figure 8. Content page A

Comparing content page A with content page B depicted in figure 9, you can see that the content pages share a layout, but that the top banner in content page B is blue instead of yellow.

Figure 9. Content page B

Simply put, while the content pages inherit the HTML layout of the master page, the content is entirely separate for each content page.

A master page comes with definition of named placeholders, although there is no requirement to replace each placeholder when a content page is created. For this reason, the master pages can be created with placeholders that contain default content. This default content will only be visible on the content page if the placeholder is *not* included in the content page. If the placeholder is included in the content page, the default content will automatically be overwritten with the custom content.

Figure 10 shows a content page where every element except Placeholder A has been replaced with blue content.

Figure 10. Default content in a content page

The person who creates a master page decides upon the name of the placeholders, as well as which placeholder contains what default content. This is important to know since each developer who is going to create SharePoint content pages needs to use the master pages created by the *Windows SharePoint Services* team when designing and creating content pages. The developer must learn what placeholders the *Windows SharePoint Services* team has defined and what content is replaceable [24].

3.1.2.2 HTTP Request Pipeline

For developers who prefer to work at a level under the productivity-centered architecture for pages and server-side controls, the *HTTP Request Pipeline* is available. The ASP.NET Framework provides the developer with control similar to the ISAPI programming model. The advantage of working with the HTTP Request Pipeline as compared to the ISAPI programming model is that creating a component for the HTTP Request Pipeline involves writing code in managed languages (such as Visual C# and Visual Basic) rather than C++. Another advantage of coding

for the HTTP Request Timeline is the availability of the APIs provided by the ASP.NET Framework. Using these APIs is much easier than using the ISAPI programming model.

The HTTP Request Pipeline contains three replaceable component types: *HttpHandler*, *HttpApplication,* and *HttpModule*. The irreplaceable fourth component *HttpContext* will be described later in this chapter. The incoming requests are enqueued and assigned to a worker thread that processes the request by interacting with each of the three component types in the HTTP Request Pipeline. Figure 11 depicts the HTTP Request Pipeline and the three replaceable components.

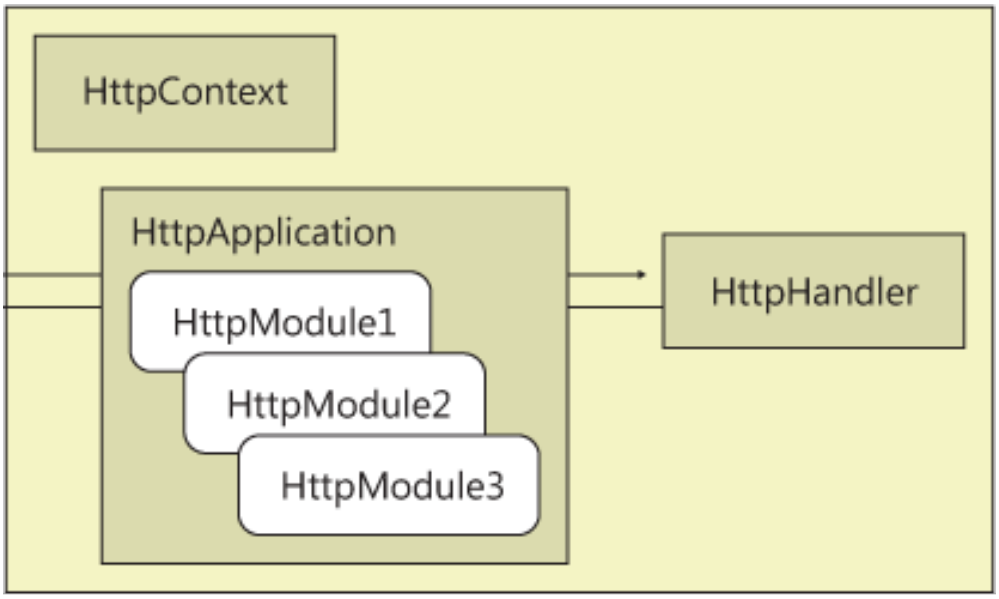

Figure 11. The HTTP Request Pipeline and its components (Used with permission from Microsoft. The figure originally appeared as figure 2-4 in [54].)

The final destination of all requests is the endpoint, which is shown in the HTTP Request Pipeline as an *HttpHandler* class. The *HttpHandler* class implements the *IHttpHandler* interface. A developer can create and plug a custom *HttpHandler* component into the HTTP Request Pipeline by adding configuration elements to the *web.config* file.

In the HTTP Request Pipeline, the *HttpApplication* component is before the *HttpHandler* component. In an application-based scenario, incoming requests are always routed through the *HttpApplication* before reaching the *HttpHandler* of the target application. This gives the *HttpApplication* the ability to pre-process all requests before being forwarding them to the target *HttpHandler*. The preprocessing stage can be divided into a series of events defined inside the *HttpApplication* class, such as *BeginRequest*, *AuthenticateRequest,* and *AuthorizeRequest*. A custom *HttpApplication*

component can be created by simply creating a file named *global.asax* and place it in the root directory of the ASP.NET application. This file defines the behavior of the preprocessing stage. If a custom *HttpApplication* component is not added, the HTTP Request Pipeline provides a default component with default behavior.

The third and last of the replaceable component types in the HTTP Request Pipeline is the *HttpModule*. The *HttpModule* and *HttpApplication* component are similar in that both are designed to handle events defined by the *HttpApplication* class. Both components are also processed before control is shifted over to the *HttpHandler* classes. A developer can for example, create a custom *HttpModule* component that handles the events *BeginRequest*, *AuthenticateRequest,* and *AuthorizeRequest*. The *HttpModule* class is defined with an interface, as with the *HttpHandler* and a custom component can be created with the *IHttpModule* interface and plugged into the HTTP Request Pipeline by adding configuration elements to the *web.config* file.

Even though an *HttpApplication* component and an *HttpModule* component work similarly, there are a few significant differences between the two. For instance, unlike the *HttpApplication* component, the *HttpModule* component is not limited to one component per application. The *web.config* file for an ASP.NET application supports the use of several different *HttpModule* components. Another difference is that *HttpModule* components can be configured on the machine level. In fact, the ASP.NET Framework comes with a set of *HttpModule* components that are automatically configured on the machine level to provide ASP.NET functionality. Examples of this functionality are Windows authentication and output caching.

The last component discussed in the HTTP Request Pipeline is the irreplaceable component *HttpContext*. When a request to send to the HTTP Request Pipeline is initialized by ASP.NET, an object from the *HttpContext* class is created and initialized with important contextual information. Viewed from the perspective of time, the *HttpContext* object is created before any custom code inside the HTTP Request Pipeline has a chance to begin execution [24].

3.1.3 Windows SharePoint Services Integration with ASP.NET

The integration of *Windows SharePoint Services* and *ASP.NET* occurs at the level of the IIS Web site. Each IIS Web site that hosts SharePoint sites must first go through a one-time transformation process in which the IIS Web site is configured to become a Web application. This transformation process consists of adding IIS metabase entries and a *web.config* file, specifically for *Microsoft SharePoint*, to the root directory of the hosting IIS Web site. Once the transformation of the IIS Web site is complete, the routing architecture of IIS and ASP.NET will be extended to properly route incoming requests through the Windows SharePoint Services runtime code.

A detailed explanation on the configuration of a *Web application* will be given in the next section of this chapter. However, before we dive into any details, it is important to understand how the concept of Web applications fits into the bigger picture of the Windows SharePoint Services architecture from the perspective of manageability and scalability.

Creation of Windows SharePoint Services Web applications are important tasks in the administration that require a certain level of administrative privileges in the web server farm. When a Web application is created, a large number of changes need to be made in the file system and the IIS metabase in every front-end web server. These changes are made automatically by the Windows SharePoint Service runtime across the front-end web servers in a Web farm environment. Fortunately, the only time this step of creating a Web application is required is when the Windows SharePoint Services are initially installed and configured.

Once a Web application is created, there will be no need to modify the file system or IIS metabase of the front-end web server when making changes in the site's collections. The architecture of Windows SharePoint Services makes it possible to establish new sites and a site collection by simply adding new entries to the configuration and content databases. This aspect of the Windows SharePoint Services architecture gives major management and provisioning advantages over ASP.NET. This manageability becomes even more important in a web server farm environment [26]. A more detailed explanation of how this is possible will be given in section 3.1.3.2.

3.1.3.1 Web Applications

There are two primary ways to create a Windows SharePoint Services Web application, using the *Central Administration Application* or the command-line function in *stsadm.exe*. As mentioned earlier, converting an existing IIS Web site on the web server can create a Web application. An alternative way is to create a Web application from scratch. In this later approach the creation of the IIS Web site occurs behind the scenes by Windows SharePoint Services. In both cases, the configuration of the hosting IIS Web site is made by Windows SharePoint Services through

addition of an IIS application map and creation of several virtual directories. Windows SharePoint Services also copies the files used in the HTTP Request Pipeline, *global.asax* and *web.config*, to the root directory of the hosting IIS Web site.

To guarantee that all incoming requests are initially routed to the ASP.NET runtime, requires adding an IIS application map to each Web application by Windows SharePoint Services. As mentioned in section 3.12, the ASP.NET runtime only registers application maps for requests targeting the well-known extensions *.aspx, .ascx*, *.ashx,* and *.asmx*. To avoid this limitation, Windows SharePoint Services configures the hosting IIS Web site with a wildcard application map to route all incoming requests i.e., not only the well-known ASP.NET extensions, but also non-ASP.NET extensions such as *.doc, .docx* and *.pdf* are routed to *aspnet_isapi.dll*.

Since all requests targeting a Web application in the SharePoint environment are routed through the ASP.NET DLL, these requests are initialized within an ASP.NET context. In the previous section, the ASP.NET HTTP Request Pipeline was discussed in detail and as mentioned, three of the component types are replaceable with custom configuration elements. The Windows SharePoint Services team utilized standard ASP.NET techniques to extend the HTTP Request Pipeline with several custom components to control the processing behavior of incoming requests. Figure 12 depicts the extended HTTP Request Pipeline are configured by Windows SharePoint Services.

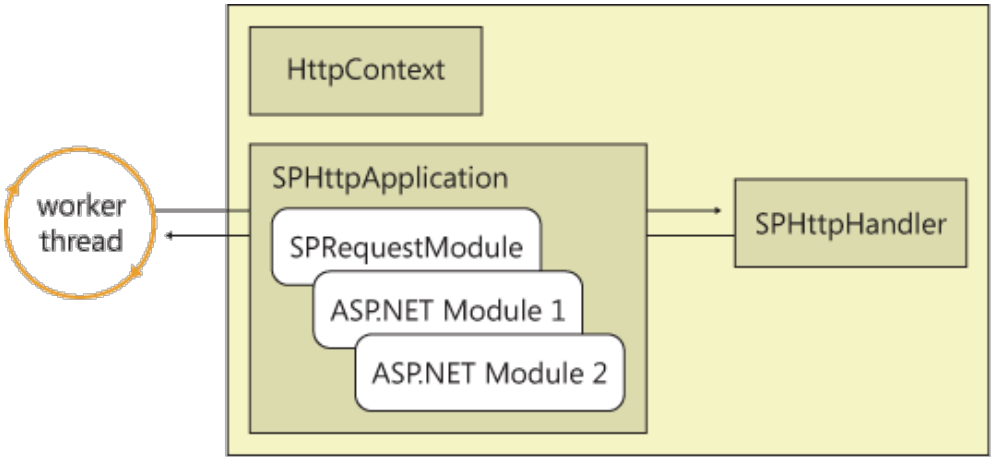

Figure 12. The extended HTTP Request Pipeline with custom components created by the Windows SharePoint Services team (Used with permission from Microsoft. The figure originally appeared as figure 2-5 on [55].)

As shown in figure 12, the HTTP Request Pipeline has been extended by the Windows SharePoint Services with a custom *HttpApplication* object for each Web application. The custom *HttpApplication* object utilizes the

SPHttpApplication class that is deployed in the Microsoft.SharePoint.dll. This class realizes the Windows SharePoint Services system assembly.

In addition to the custom *HttpApplication* component, the Windows SharePoint Services architecture also integrates a custom *HttpModule* and a custom *HttpHandler* component into the HTTP Request Pipeline.

SPRequestModule, the custom *HttpModule* created by the Windows SharePoint Services team initializes various features of the SharePoint Services runtime environment. By default in the Windows SharePoint Services *web.config* file, the *SPRequestModule* is the first *HttpModule* that responds to events occurring on the application-level in the HTTP Request Pipeline in ASP.NET. Although the default *web.config* file was replaced by Windows SharePoint Services, several of the standard *HttpModule* components that come with the ASP.NET Framework remain in the new *web.config* file. For instance, the components that deal with output caching and different types of authentication are useful for Windows SharePoint Services.

The last custom component *SPHttpHandler* created by the Windows SharePoint Services team is configured to be the single endpoint for all incoming requests. By extending the HTTP Request Pipeline, Windows SharePoint Services has full control over the fundamental capabilities of the ASP.NET Framework as well as every incoming request targeting a Web application [26].

3.1.3.2 SPVirtualPathProvider

A major strength of Windows SharePoint Services running over ASP.NET is the ability to create and customize pages within a site without altering anything in the local file system of the front-end web server. This functionality is made possible by storing the customized versions of the physical *.aspx* and .master files in the content database. These entries in the content database are retrieved when a request targeting this page is received. The architectural details that make this possible will be explained further later in this chapter.

Page customization in Windows SharePoint Services works by storing customizations in the content database. Consider a simple example where modification of the HTML layout in the home page (*default.aspx*) is done using *Microsoft Office SharePoint Designer*. When saving a page using SharePoint Designer, the entire contents of this customized page definition is written to the content database by Windows SharePoint Services. Once the data has been stored in the content database, Windows SharePoint Services retrieves the contents of this customized page definition when a request for this page arrives. The retrieved content will be passed to the ASP.NET runtime for parsing.

A critical component that makes this possible is the *virtual path provider*, a new type of pluggable component introduced in ASP.NET 2.0.

The purpose of a virtual path provider is to hide the details of where page files are stored from the ASP.NET runtime. A developer can create a virtual path provider and design a custom component that retrieves the content of ASP.NET file types from a remote location. This content can then be passed along to the ASP.NET runtime for parsing without divulging the details of where the physical file is located.

The virtual path provider *SPVirtualPathProvider* was created by the Windows SharePoint Services team and integrated into the request-handling infrastructure in the ASP.NET Web application through the *SPRequestModule*. A Web application is initialized by the *SPRequestModule* component, which contains the code to register the *SPVirtualPathProvider* class with the ASP.NET Framework. The role which *SPVirtualPathProvider* plays in the Windows SharePoint architecture is shown in figure 13.

As shown in figure 13, an ASP.NET file (shown as "default.aspx") is retrieved from the content database by the *SPVirtualPathProvider* and then passed to the ASP.NET page parser. The ASP.NET page parser receives information about how the page should be parsed from a class named *SPPageParserFilter*. This parser filter class collaborates with the *SPVirtualPathProvider*. The *SPPageParserFilter* component controls how the retrieved ASP.NET file should be processed, for example if it should be compiled into a DLL or processed without being compiled.

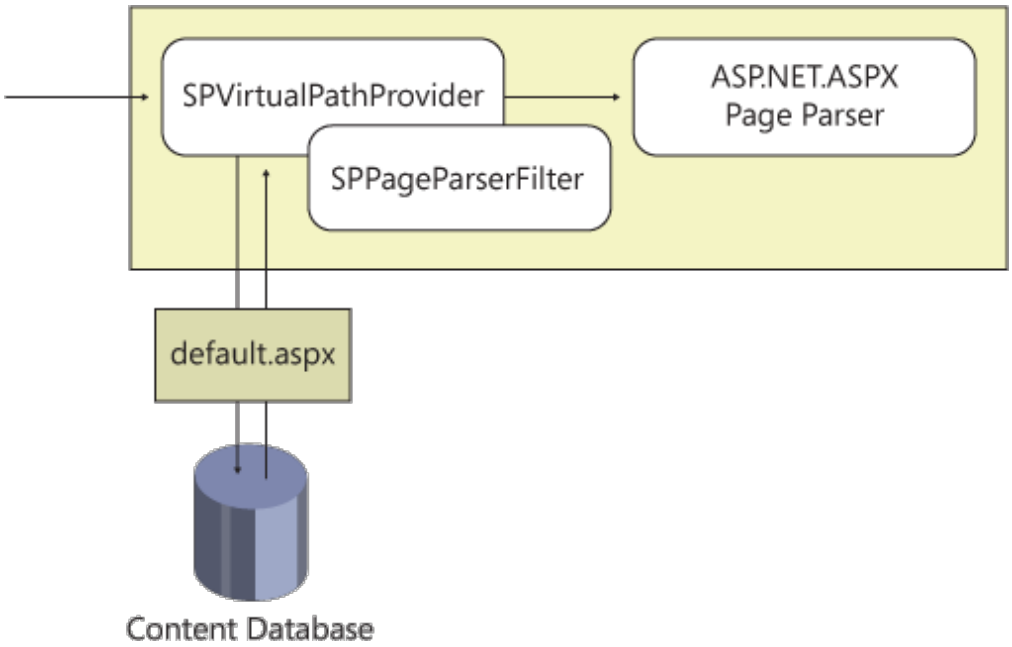

Figure 13. SPVirtualPathProvider's role in Windows SharePoint Services (Used with permission from Microsoft. The figure originally appeared as figure 2-6 in [56].)

The *SPVirtualPathProvider* provides the foundation that supports page customization in the Windows SharePoint Services architecture. Additionally, it supports another key feature that optimizes the scalability of the Windows SharePoint Services architecture: *page ghosting*. With *page ghosting*, a server farm is able to scale out to thousands of pages across all sites. These two optimizations provided by the *SPVirtualPathProvider*, *page customization* and *page ghosting*, are both key factors in the scalability of Windows SharePoint Services.

Consider a scenario where 100 new SharePoint sites are created using the *Blank Site* template. The Blank Site template is used for creating blank home pages [28]. These 100 sites across the farm are identical and none require a customized version of the home page *default.aspx*. In this case, copying the exact same page definition file into the content database a hundred times is impractical and redundant. This can be avoided with *page ghosting*. Since pages such as *default.aspx* are based on page templates that reside in the file system of a front-end web server, Windows SharePoint Services simply provisions an instance of the un-customized page based on the *default.aspx* page template *when requested*. Instead of storing 100 copies of *default.aspx* in the content database, Windows SharePoint Services utilizes the same page template as needed. A page template is compiled into an assembly DLL that only needs to be loaded into a Web application once by the IIS worker process during initialization. *Page ghosting* provisions a page instance by processing a page template located in the file system of the front-end web server.

Unfortunately, a modified page eliminates the possibility of *page ghosting*. Instead, the *SPVirtualPathProvider* retrieves the customized version of the page from the content database. In Windows SharePoint terminology customized pages are referred to *un-ghosted.*

The *SPVirtualPathProvider* plays an important role in the Windows SharePoint architecture because it determines whether a page has been customized, and whether a page should be processed as *ghosted* or *un-ghosted*. Furthermore, the details of *ghosted* and *un-ghosted* pages are withheld from the ASP.NET runtime, which is a valuable aspect of Windows SharePoint Services [26].

3.1.3.2.1 Page Parsing in Windows SharePoint Services 2.0

The older versions of Windows SharePoint Services were based on ASP.NET 1.1. However, the previous version of ASP.NET did not have any equivalent to the virtual path provider model. The Windows SharePoint Services team solved that problem by creating their own *.aspx* page parser for parsing un-ghosted pages after retrieving them from the content database. Compared to the current ASP.NET page parser, the *.aspx page parser* created by the Windows SharePoint Services team was simpler and did not offer as many features, such as hosting user controls. The new architecture in Windows SharePoint Services 3.0 introduced, among other

things, the *SPVirtualPathProvider* and the ASP.NET page parser. These two features should be considered the most significant architectural improvements over the previous version, Windows SharePoint Services 2.0 [26].

3.1.3.3 Advantages and disadvantages of Ghosted & Un-Ghosted Pages

Earlier in this chapter, the *SPVirtualPathProvider* component and the principles of page *ghosting* and *un-ghosting* were introduced. The concept of *page ghosting* is an optimization used to enhance the scalability of page rendering and processing and has obvious advantages for scalability within a web server farm. Another advantage of the *SPVirtualPathProvider* component is the flexibility of page customization. When a user uses Microsoft Office SharePoint Designer to customize a site page, a customized version of the page definition is saved and stored in the content database. Unfortunately, this flexibility in customization can have a negative impact on performance and scalability. When a request for an *un-ghosted page* arrives, the *SPVirtualPathProvider* must first retrieve the page definition from the backend database server before passing it on to the Microsoft ASP.NET page parser. The ASP.NET page parser must then parse and load the page definition into memory before it can process the page and return content to the user. Since every un-ghosted page definition must be separately parsed and loaded into memory within the Web application's application pool, a Web application with thousands of un-ghosted pages requires more memory than a Web application with only a hundred un-ghosted pages.

Un-ghosted pages are not processed and compiled into an assembly DLL using the standard ASP.NET model, but instead are parsed by the ASP.NET page parser before being processed using the no-compile mode feature that was introduced with ASP.NET 2.0. The reason why an *.aspx* page is parsed in no-compile mode is because this can be more efficient and scalable in certain scenarios, such as large Windows SharePoint Services environments where the number of un-ghosted pages can reach up thousands or tens of thousands.

No-compile pages have an advantage over compiled pages since the Microsoft .NET Framework does not support unloading assembly DLLs from memory. The closest equivalent to this process would be to recycle the current Windows process or .NET *AppDomain* class. Unfortunately, recycling involves unloading *all* DLLs from memory since there is no ability to unload only those DLLs that have not been recently used. Moreover, there is an upper limit on the number of assembly DLLs that can be loaded into a .NET *AppDomain*.

Higher levels of scalability can be reached with no-compile pages since there is no need to load new assembly DLLs or managed classes into memory. Instead, the loading process with no-compile pages deals with

control trees, which are more manageable for Windows SharePoint Services than assembly DLLs. For instance, when Windows SharePoint Services has finished processing an un-ghosted page, it can free up the memory by unloading the page's control tree. Another advantage of no-compile pages is that it eliminates the compilation process. However, the disadvantage is that if the page is accessed again it has to be reprocessed rather than simply using the cached compiled version. So there is a question of what fraction of pages are only accessed once. [38].

3.1.3.4 Virtual Directories Within a Web Application

During the conversion from an IIS Web site to a Web application, several virtual directories, including the *_controltemplates* directory, the *_vti_bin* directory, the *_wpresources* directory, and the *_layouts* directory, are created by the Windows SharePoint Services. Figure 11 shows how the virtual directories in a Web application can be examined using the IIS Manager tool.

As shown in figure 14, the virtual directories are each mapped to a physical directory on the file system in the path *C:\Program Files\Common Files\Microsoft Shared\web server Extensions*. Several aspects of the Windows SharePoint Services runtime use these virtual directories [26].

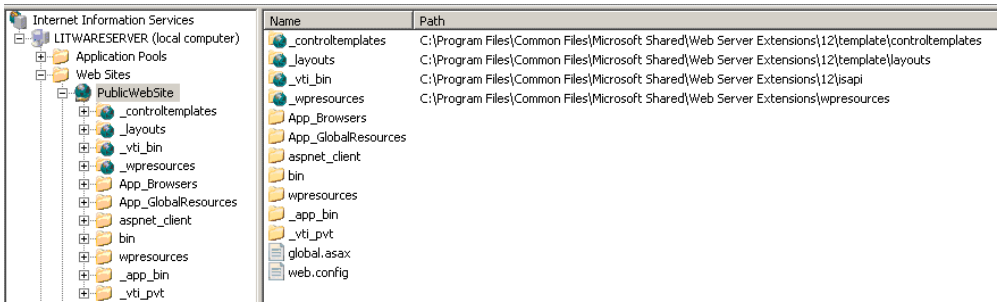

Figure 14. The virtual directories observed in the IIS Manager tool (Used with permission from Microsoft. The figure originally appeared as figure 2-7 in [57].)

3.2 Microsoft SharePoint Topology

A detailed description of the foundation for Windows SharePoint Services and how it operates behind the scenes was given in the previous section. This section will discuss the top level of Windows SharePoint Services. It will briefly touch on the subjects of Web applications and the concept of site collections and sites. Following this is a description of a specific site collection relevant to this thesis: My Site. The last part of this chapter will focus on other parts of the SharePoint topology, such as central administration, administration & security, and managed metadata.

3.2.1 Web Applications

Web applications are the content containers at the top level of a SharePoint farm, and generally the interface that a user utilizes for interaction with SharePoint. Web applications are independent of each other and can be restarted independently in the IIS application pool. As mentioned earlier, Web applications are IIS Web sites that have been created and configured as Web applications. The application maps and URLs associated with Web applications are defined via the SharePoint central management console, then replicated into the IIS configuration of every server in the farm [1].

3.2.1.1 Sharing and Isolation

A unique domain name can be assigned to each Web application. The use of unique domain names isolates this Web application from other Web applications and helps to prevent cross-site scripting attacks [30].

3.2.1.2 Configurable items

There are three configurable items or settings that contribute to the isolation and sharing in Web applications; *Service applications*, *zones,* and *policy for Web applications* [30].

Service applications are services deployed on a farm. Service applications provide resources that can be shared across sites within the farm, or in some cases across multiple farms. To prevent the farm from discontinuing operation in the event of a Service application failure, Service applications are specifically designed to be as independent as possible. Each Service application usually has its own configuration database and Active Directory service account [17]. The CPU usage varies depending on the Service application. Since SharePoint has to handle all user requests coming from every enabled Web application and Service application in the farm, CPU usage may reach 100% at times [18]. The search indexing Service application, for example, can use 100% of RAM depending on the indexing interval and the amount of data stored in the farm.

Zones are used when the administrator wishes to enforce different access and policy conditions on a large group of users. Zones are realized by using different URLs to access to the same Web application, but they represent different Web sites in IIS. A Web application can be extended into five different zones, each using one of the available zone names: *Default*, *Intranet*, *Internet*, *Custom,* or *Extranet*. When a Web application is created, it is created with the Default zone. Extending the Web application can create other zones. Zones with the same named zones are generally coordinated and configured to be used by the same group of users. Each zone can be configured to use a separate authentication provider. Zones enable users to share content across partner companies [30].

A *policy for Web applications* allows the administrator to enforce specific permissions on all content across one or more zones in the Web application. This enables the administrator to set security policies for users at the Web application level, and the permissions in this policy overrides every other security settings that has been set for sites and content [30].

3.2.2 Site Collections

A set of Web sites that share the same owner and administration settings are called a *site collection* in Windows SharePoint terminology [25]. Site collections can be created and deployed either with or without hostnames, although the first option is preferred in SharePoint 2013 [27]. Hostnames simplify access to servers and sites for users by mapping an IP address to a human-readable label commonly made up of letters and words [58]. For example the deployment used in this thesis; Instead of accessing the My Site site collection with an IP address on the Web farm servers, users can access their personal sites by simply entering *http://sp2013/my/* on the address field. The address *http://sp2013/my/* is an example of a hostname.

Site collection administrators can upgrade individual site collections to enable new features on SharePoint Online 2013. An upgrade also makes user interface improvements available on the site collection.

3.2.2.1 Capacity

The recommended maximum number of site collections implemented per content database is fewer than 50,000 due to the limited availability of ports for TCP/IP connections on a system. This recommendation was specified to ensure acceptable performance, although the performance can degrade at around 10,000 site collections. In order to provide additional storage capacity and throughput, the site collections can be scaled out and distributed across multiple database servers [27].

3.2.2.2 Sharing and isolation

Site collections introduce various sharing and isolation prospects as they allow different levels of control over site features and settings [25,26].

Items that are stored in file systems, for example features in the virtual directory *_layouts*, can be shared across site collections. However, there are items that can only be shared within a site collection, such as [27]:

- Master pages,
- Page layouts,
- Images, and
- Site templates.

Apart from file isolation, permissions and navigations are also isolated in a site collection in the following ways:

- Permissions are inherited from the top-level site.
- A site collection cannot inherit permissions from another site collection.
- Built-in navigation between site collections is not available.

From SharePoint Server 2010 on, regardless of the number of site collections or databases, search results are aggregated across site collections. Depending on the user, the search is aggregated over the site collections the user has permissions to.

However, even though permissions are applied to each individual site, cross-site scripting attacks may still occur between sites within the domain [27]. Fortunately, this problem can be prevented by installing a patch provided by Microsoft [59].

3.2.2.3 Configurable items

There are a few items in both Windows SharePoint Server and site collections that contributes to isolation and sharing. Several items in Windows SharePoint Server contribute to both isolation and sharing. These items are the *site collection administrator*, *site template,* and *quota template*.

A *primary and secondary administrator* can be set for the site collections in the farm. These administrators hold the administrative permissions to every site collection in the farm.

A *site template* determines which lists and features will be available on the new site. The site template is not editable after site creation, although many parts of the site can be customized.

The *quota template* can be applied to site collections to limit how much storage can be used for the site collection. There are two templates provided by Windows SharePoint Server: a quota template of 100 MB for a Personal Site and a quota template of 2000 MB for Team Sites.

Additionally, there are two configurable items within a site collection that also contribute to isolation and sharing; *Site collection administrators* and *permission level*. These items become available after a site collection has been created with the settings mentioned above.

Site collection administrators hold the permissions to administer the site collection, and multiple users can be specified as administrators. Unlike user accounts, group accounts cannot be specified as administrators, although they can be added to permit access to site collections. Separate *permission levels* can be set for both user and group accounts [27].

3.2.2.4 Administration

Since DNS entries are only required when creating host-named site collections, the task of creating site collections can easily be automated or

delegated to users in the farm. The team sites can be created either centrally in using *Central Administration* or by activating the *Self-Service Site Management* feature. *The Self-Service Site Management* feature allows users to create their own site collections, and the *My Site* site collection is an example of this [27].

3.2.3 Sites

A site is hosted inside a site collection and consists of one or more Web pages and other items such as lists, documents, and libraries [25]. Typically, a site has a home page, a starting point which users initially request. The home page is interconnected with other pages in the site via hyperlinks. Sites utilize a *top-level hierarchy*, with a Web site at the top. The top-level Web site can contain one or more subsites, which in turn can contain more subsites [31].

The use of a top-level site and subsites makes it possible to divide the site content into separately manageable sites. This hierarchy makes it possible to divide a site into a main working site for a team, with separate individual working sites. The top-level hierarchy also allows an administrator to set different levels of control on a site's features and settings [26].

3.2.3.1 Capacity

A guideline to maintain acceptable performance is to create less than 250 000 sites per site collection due to the limited availability of IP addresses. It is possible to create a very large number of Web sites by nesting subsites. However, this is undesirable since a large number of nested subsites can greatly increase the time needed for an upgrade. Microsoft recommends having at most 5000 sites within a site collection to maintain a good operational state [31].

3.2.3.2 Sharing and isolation

Unlike site collections where there is no built-in navigation between site collections, sites include built-in navigation between subsites within the site collection.

Just as for site collections, separate sites are also vulnerable to cross-site scripting attacks from other sites within the same domain [31].

3.2.4 My Site

My Site is a Web application that consists of a host site collection, an individual site collection, and several *Service Applications* and features. *My Site* is a part of the *User Profile service application*, and is enabled by default by Windows SharePoint Server. The individual site collection is the only part of the *My Site* infrastructure that a user may configure. The

remaining parts of the infrastructure are configured once and shared by all users within the organization [16].

Each user has their own individual site collection in the *My Site* Web application. The individual site collection can be configured and personalized. Social networking was introduced in Microsoft SharePoint 2007, and as mentioned in the introduction chapter, Microsoft added various capabilities for social networking. Storage space for a user's personal documents is available under *SkyDrive Pro*, and the link *Sites* in the personal home page enable quick access to Workspaces within the web server farm. The *My Site* Web application also offers Microsoft Office integration [32]. Figure 15 shows an example of a *My Site* home page.

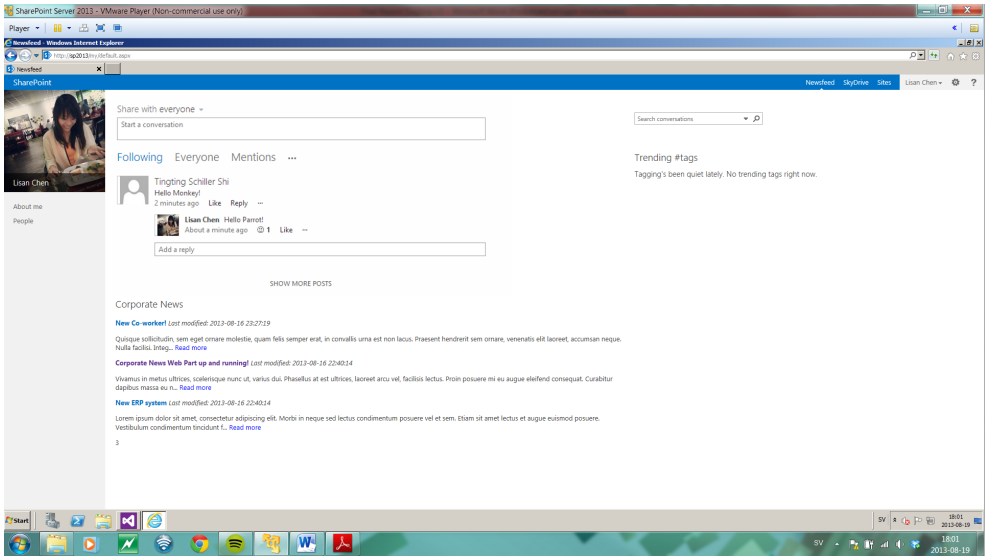

Figure 15. An example of a My Site home page

The My Site Web application was improved in Microsoft SharePoint 2007 with the addition of Privacy Controls that enable users to toggle permissions between three different levels; *Managers*, *Colleagues,* and *Everyone* [33].

Although a user cannot customize the home page of their *My Site*, the administrator is able to customize and add Web parts to the default layout of a My Site home page. When a Web part is added to the default layout, the Web part will be visible for all users in their individual home pages in My Site [34].

3.2.4.1 My Site Architecture

A *My Site* host is automatically created when a *Shared Services Provider* (SSP) is created. The location of the site collection is created in the file system of the Web application, and all personal sites are stored in that location. In most scenarios, the *My Site* site collection generally only has

one location per deployment. An individual *My Site* is created if the user profile information is available in the *Active Directory Directory Services* and imported by the SSP [35].

In general, a user is limited to only one SSP, in other words one physical My Site location. However, there are exceptions, for example when a *My Site* deployment is geographically distributed, then a dedicated SSP is used for each region. It is challenging to provide a predictable experience for users in the case of a geographically distributed deployment with multiple My Site locations and multiple SSPs. A solution to this is to use *global deployments* with trusted locations where user profile information can be replicated. Such a global deployment results in one set of personalized content for each user being available across all locations and SSPs [36].

Global deployments essentially mean *multiple* deployments of SSPs. In this approach, My Site is deployed across multiple SSPs. A global deployment may be a preferred solution in some cases, for example when there is poor network connectivity between a remote set of users and the primary SSP. However, there are a few disadvantages to global deployments in the form of lost functionality with respect to personalized links and searching for people based on social distance. It is recommended to have a good replication solution ready before enabling a global deployment as this replication solution can save time [37].

3.2.5 SharePoint Farms

If a SharePoint server topology consist of more than one server that share common resources, then it is generally known as a SharePoint farm. Although a SharePoint farm normally runs independently, it is also possible to provide and subscribe to functionality in another farm [15]. The administrative settings and configurations of each farm are made in a central configuration database. This database can be managed through Microsoft's task automation framework, Windows PowerShell, or through a Central Administration website. Each server in the topology must directly communicate with the central configuration database to configure its services to match the requirements of the farm and to report issues regarding server health, resource allocation, etc.

3.2.6 Central Administration

Central Administration is another default Web application in the SharePoint Foundation. It is deployed at a central location and used to perform administrative tasks in SharePoint. The Central Administration can be divided into four different parts: Home page, Operations page, Application Management page, and Shared Services Providers page [17]. The functions of these parts are:

- The *Home page* contains a list of tasks the administrator should complete, a complete list of servers deployed in the farm topology, and complete list of available resources.
- The Operations page contains links to pages where the administrative tasks are performed, such as Topology and Services, Security Configuration, etc.
- The Application Management page contains links to pages where an administrator can create and manage Web applications and Site Collections.
- The Shared Services Providers page contains a list of links to SSPs. Clicking one of these links enables the administrator to access the SSP's administration home page. An example of SSP is: User Profiles and My Sites.

3.2.7 Administration and Security

The security approach taken in SharePoint's architecture is the "least-privileges" execution permission practice. The main idea is to avoid giving unnecessary privileges by granting just enough permission to users without compromising the security in the Web farm environment [19].

Because SharePoint utilizes a centralized management system, the Central Administration usually runs on a single server, although it is possible to deploy Central Administration on several servers for redundancy [8]. The Web application provides an interface for other Web applications and Service applications in the farm; it also provides a management interface for Active Directory accounts. If the Central Administration goes down, the farm can, usually be reconfigured on the Central Administration server by using Windows PowerShell.

3.2.8 Managed Metadata

This section will present Managed Metadata in SharePoint, as well as benefits of using managed metadata.

3.2.8.1 What is Managed Metadata?

Managed metadata is a hierarchical collection of defined terms that can be used as attributes for items in SharePoint lists or libraries. A term is a word that can be associated with an item in a SharePoint list or library. A term set is a collection of terms. A SharePoint list column can be configured to only contain terms from a specific term set [40].

Global term sets are created and stored in the Managed Metadata Service (MMS) application and they are available for all site collections

connected to a MMS instance. Local term sets are created within a site collection when creating a site column in a SharePoint list or library. Unlike global term sets, local term sets are only available to the specific site collection that contains the added site column [41]. Figure 16 illustrates how to bind a column to the different term sets.

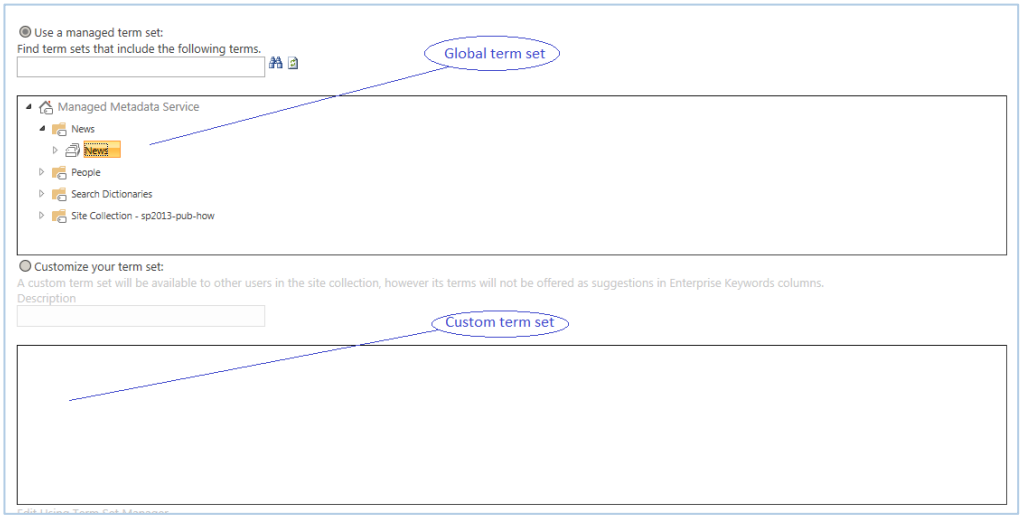

Figure 16. Global term set and custom term set. Users can manually add terms to the empty custom local term set.

There are two types of terms:

- Managed terms are predefined by authorized users and they are often organized into a hierarchy.
- Enterprise keywords are words that have been added to SharePoint list items and that are not selected from any predefined term sets. Enterprise keywords are a nonhierarchical term set called the keyword set.

Both types of terms are stored in the Managed term store, as shown in Figure 17 [40].

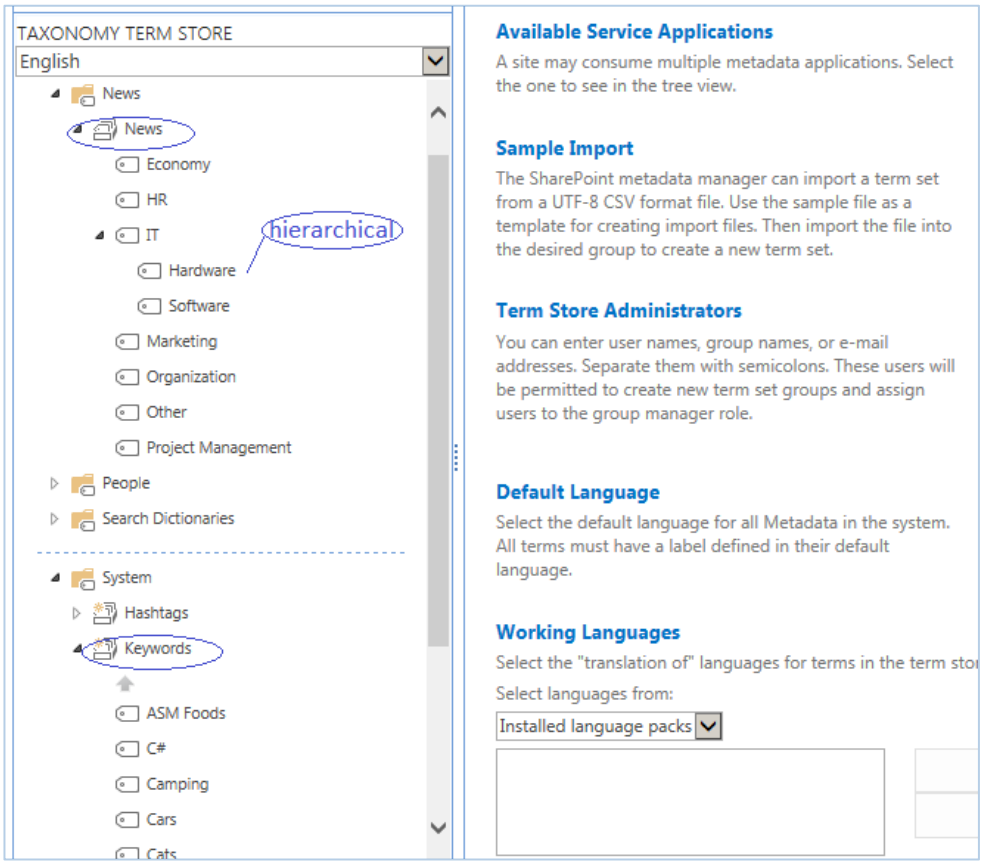

Figure 17. Term set and Enterprise keyword set.

3.2.8.2 Benefits of using Managed metadata

According to Microsoft, there are several benefits of using Managed metadata [40]:

- Consistent use of terminology. Using Managed metadata gives more consistent use of terms and enterprise keywords that are added to the SharePoint items. Terms can be predefined and they can be created by users or only created by authorized users based on system settings. Authorized users can also forbid users from adding their own keywords to items and require them to choose from existing enterprise keywords. By presenting only term sets or enterprise keywords set from which users can choose values, Managed metadata provides better accuracy and a higher degree of confidence that the provided data is correct and always valid.
- More relevant SharePoint search results. When items have consistent attributes chosen from the existing term

sets and enterprise keywords set, the search will provide more accurate results.

- Dynamic metadata behaviour. When adding a new term to a term set, all columns that map to the term set are updated with the new set of values. Using terms therefore helps to synchronise SharePoint items with changes in business. For example, a term set contains a list of terms that represents the names of different products. When one of these product names needs to be changed, one should simply edit the value of the product's term and it will automatically apply to all columns that are mapped to the term set.

4 Method

This chapter will present the tools and the methods used for this thesis project and the methods used during the implementation of the web part.

4.1 Tools

The tools that were used during the thesis were:

- SharePoint Server 2013 VMware Player
- Microsoft Visual Studio Ultimate 2012

The SharePoint Server was used for administration and configuration of the SharePoint service applications and sites. Visual Studio was used for development of the web part. The company for which the students did this thesis project had the above tools installed and ready for use when the thesis project started.

4.2 Project method

During this thesis project, a light version of the agile method Scrum was used to provide good organization and communication among the team members (in this case the two students who are the authors of this thesis). Tasks were prioritized and divided into small parts for the Scrum board (see Figure 18). The board was updated weekly and the students made summaries of the achievements and coming goals every two weeks. The programming part was mostly done individually as the implementation was divided into two parts. These two parts were User part and Company part. The User part was implemented by Lisan Chen and the Company part was implemented by Tingting Schiller Shi.

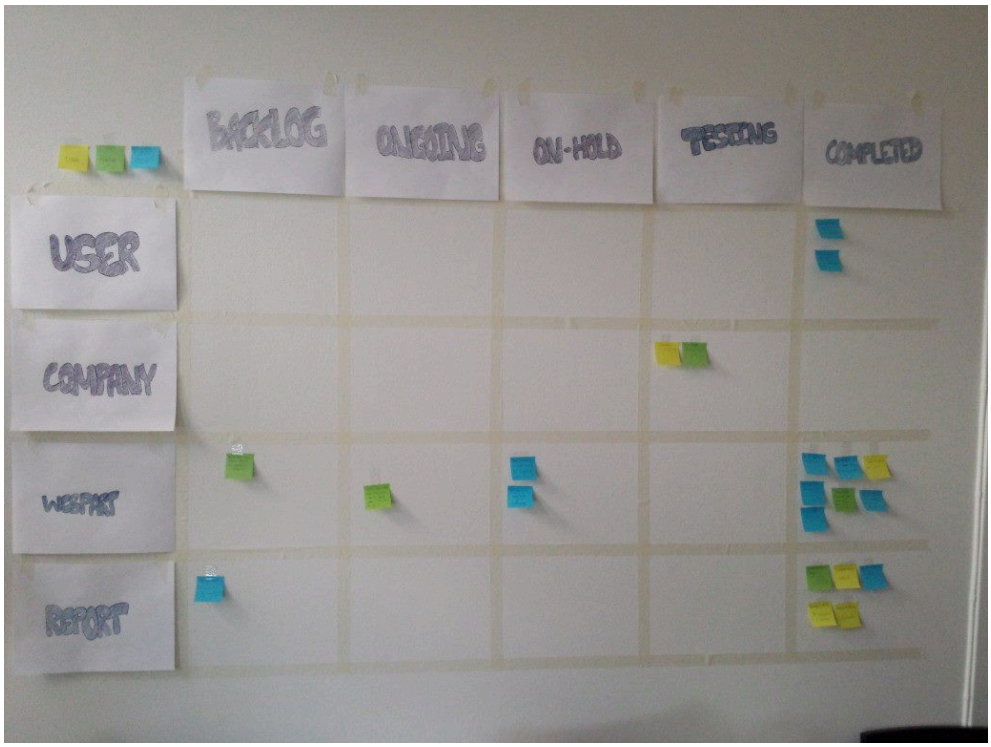

Figure 18. Scrum board

4.3 Implementation of Corporate News Web Part

The goal of the thesis was to implement a web part that retrieves and shows an overview of corporate news that matches the user's subscribed news categories. To do this, a *Corporate News Web Part* was created and added to the default page of *My Site*. The implementation of the web part was divided into two parts: User part and Company part. The main objective on the user side was to enable users to select the news categories they want to subscribe to, while the company side focused on retrieving news contents relevant to the user's subscriptions. The following sections will give a more detailed description of each step in the development of these two parts.

4.3.1 Creating and adding a web part

Since the parts were expected to operate in a SharePoint 2013 environment, the web part must be created as a SharePoint 2013 project. The reason is that many features in SharePoint 2010 have been deprecated in the later version. As a result, things that worked in the previous version may not behave properly in the new one [42]. One problem occurred during the creation of the project. It turned out that there was no option for creating SharePoint 2013 projects in Visual Studio 2012. To solve this, the Microsoft

Office Developer Tools for Visual Studio 2012 were installed in order to extract the relevant project templates for use in Visual Studio [43]. Hence the group *Office/SharePoint* was added to the *New Project* menu, in which a SharePoint 2013 projects options now appeared.

As illustrated in Figure 19, the web part was created as a SharePoint 2013 Visual Web Part. The site used for debugging was set to *http://sp2013/my* which is the user's *My Site* page where the web part will be added. Also the option *Deploy as a farm solution* was chosen over the *sandbox solution*, since a sandbox solution has only a limited set of tools that can be used. Unlike a farm solution, a sandbox solution is only deployed with a specific site collection instead of for a whole farm; it also lacks access to some file systems and namespaces that are useful for developing web parts [44].

To add the created web part to My Site, the project was deployed and added to *default.aspx* which is the content page linked to *mysite.master*. This also the default *My Site* page which contains the news feed.

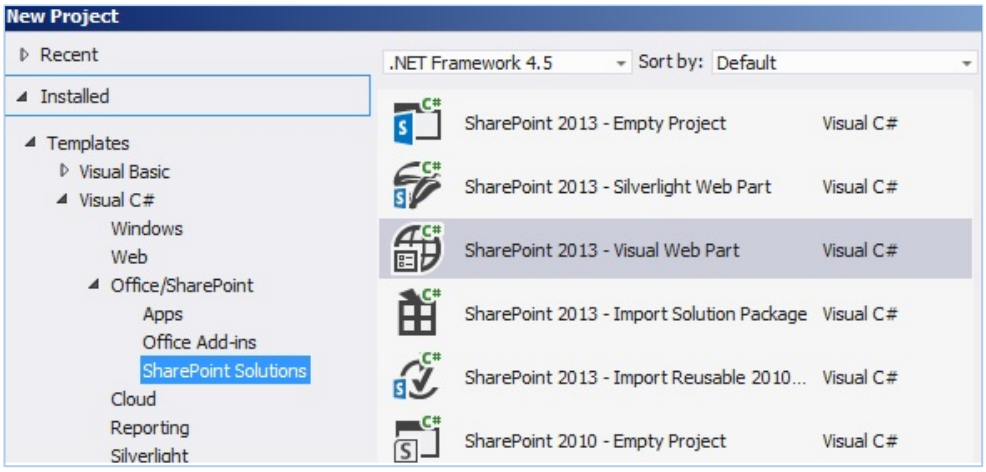

Figure 19. Creating Visual Web Part in Visual Studio

4.3.2 User side

The user side of the web part implementation consisted of one administrative part and one coding part. The objective of the administrative part was to enable users to subscribe to news categories via their user profile in *My Site*. The coding part focused on retrieving the user defined news categories from the users' *My Site*.

4.3.2.1 Administration

In SharePoint user profiles, there is a default set of user properties which provide information about a user. Administrators can add custom properties to user profiles via *CA* in order to store additional information about the

users [45]. One objective of this thesis project was to allow users to select news categories to subscribe to. This was achieved by creating a custom property *News* using the *CA*, adding the property to a section in the user profile, and binding this property to a collection of news categories from which a user is allowed to select values. The collection of news categories is a hierarchical term set that consists of terms that represent the different categories. Figure 20 shows how to bind a user property to a term set. A more complete description of the term set will be presented in section 4.3.3.1.

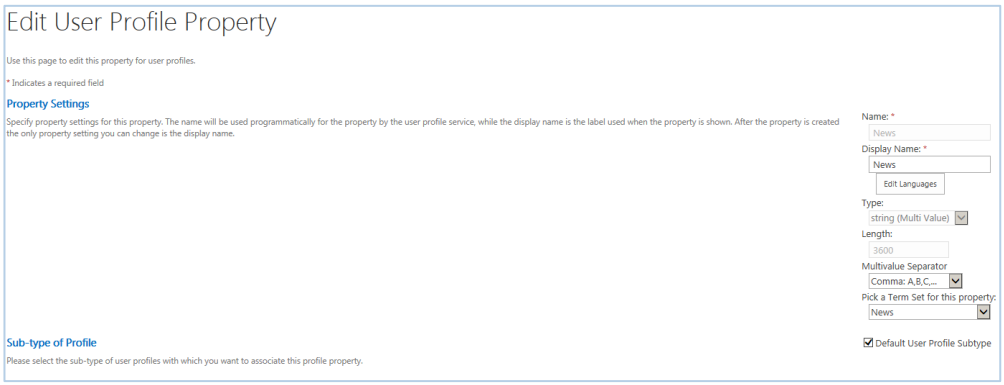

Figure 20. Configuring custom user property using CA.

4.3.2.2 Implementation of web part

Once the *News* property was added, users can select those news categories they want to subscribe to in their user's profile. The web part also needs to be able to automatically retrieve the values set in this property. The property values were retrieved using the following classes:

- *SPSite* was set to *http://sp2013/my* which gets the SPSite object that represents the My Site site collection.
- *SPServiceContext* was used to get the service context of the site.
- *UserProfileManager* was used to initialize a user profile config manager object by getting the current logged in user's id using *SPContext*.
- *ProfileValueCollection* was used to get the *News* property values for the logged in user.

4.3.3 Company side

Similar to the user side, the company side also consisted of an administrative part and a coding part. The administrative part focused on

organizing of the published news and the coding part focused on selecting news contents relevant for the user's subscriptions.

4.3.3.1 Administration

When publishing news contents, the publisher chooses the news categories that they wish to add to each news item. The categories should also be accessible from the user side in order to allow users to select which categories they wish to subscribe to. To organize the news articles, a centralized collection of news categories was created in the *Managed Metadata Service* application as a term set named *News*. This term set contains a list of terms that represents news categories, such as *Economy*, *IT,* etc. As the term set is hierarchical, sub terms can be added to each term. As illustrated in Figure 21, the term *IT* has two subcategories, *Software* and *Hardware*, which are the sub terms of the parent term IT. This way the categories are organized in a centralized manner and the term set can be accessed from both user side and company side. However, only authorized users can add and edit terms in the term set.

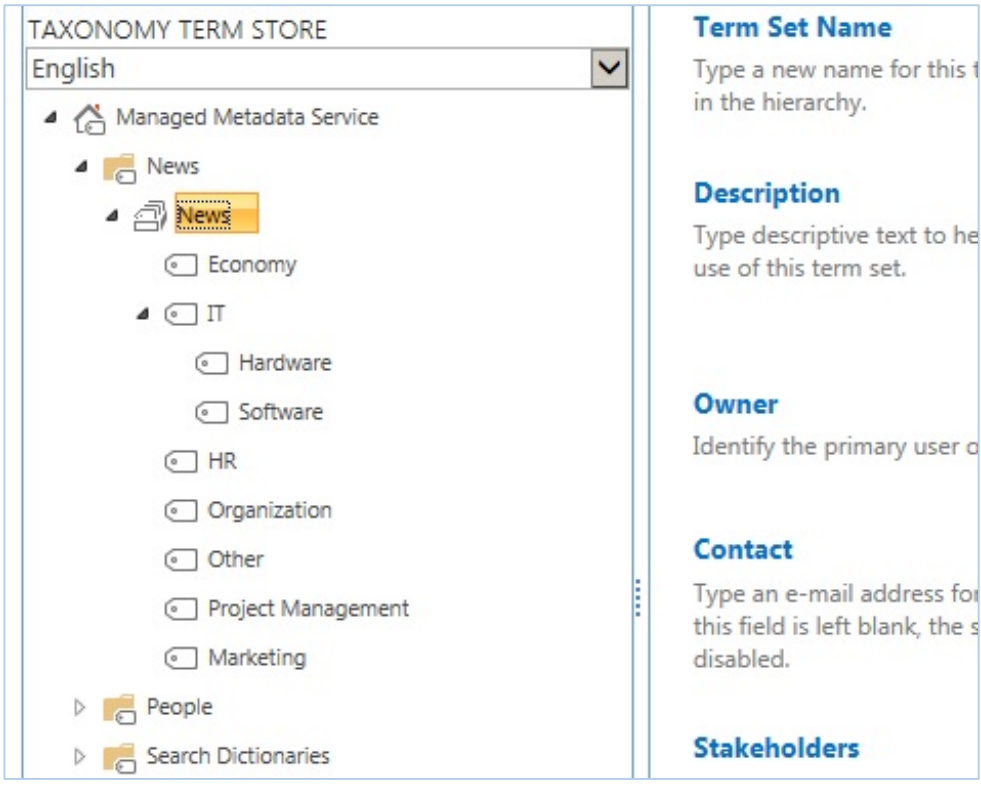

Figure 21. *News* term set

Once the *News* term set was ready, it was possible to add news items with these different categories. For this purpose, a custom site column was added to the SharePoint library where the news articles were stored, in this

case the *Pages* library (shown in Figure 22). This library was set to be available to other sites and site collections through search. The site column type was set to Managed Metadata and required the use of the *News* term set from which the publisher selects the applicable news categories. *Content crawling* is necessary to make contents searchable in SharePoint, hence a full crawl was done to make the new site column searchable. This crawling was done using the *Search Service Application* in *CA*.

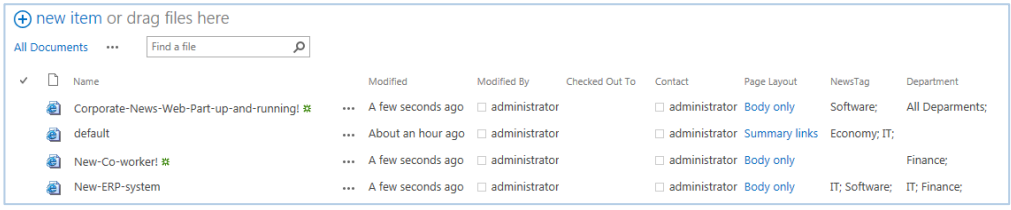

Figure 22. Pages library with custom site columns: *NewsTag* and *Department*

Similar to the *News* term set, another term set *Department* was created. As the name indicates the term set contains a collection of departments within the company. A custom metadata site column bound to the *Department* term set was added to the *Pages* library for news articles (see Figure 23). The column was made searchable after a full crawl. A news publisher is now able to tag new articles with department names.

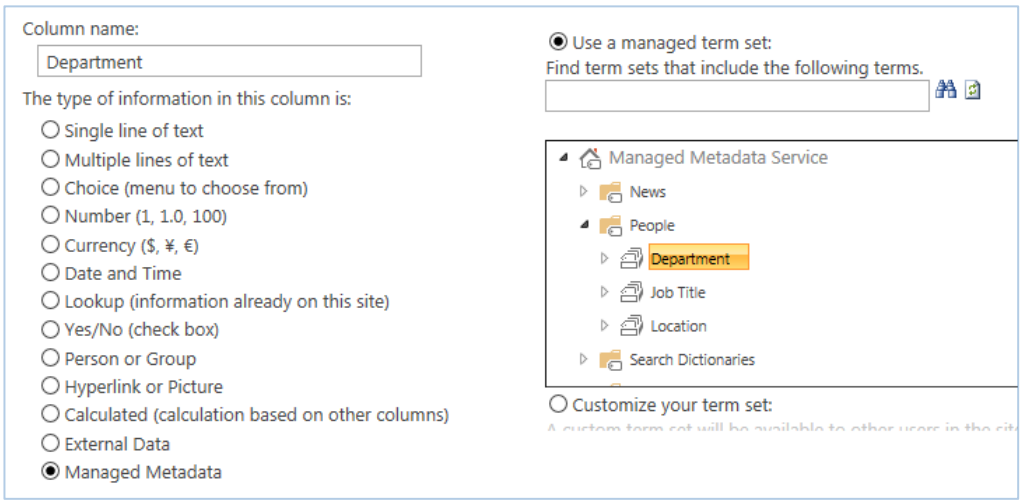

Figure 23. Creating site column and binding it with global term set

4.3.3.2 Implementation of web part

As mentioned in section 4.3.2, the web part is able to get the values stored in the user's *News* property. The purpose of the web part is to select news content that matches the retrieved user values. To get news content

from the *Page*s library, the *SPWeb* and *SPSite* classes were used to get the *SPSite object* that represents the current site collection, which in this case represented the site collection *Corporate News* (where the news articles were stored). The properties in the *SPSiteDataQuery* class were configured as follows:

- The *Webs* property defines the scope, e.g. which web sites to include in the query. In this case, the scope was set to be Recursive, which includes the current web site (Corporate News) and all its sub sites.
- The *List* property defines the lists and libraries to include in the query. The lists template ID 850 was specified as this is the template id for Pages libraries.
- The *ViewFields* property specifies the list of fields to return, in this case the Title, Modified, and Page Content fields. Note that the specified fields must be given as the internal field name rather than the displayed field name. The internal field names are found in the library settings page of the Pages library.
- The *Query* property selects and sorts the relevant data to return. The selected data was set to be news articles tagged with news categories that matched the user's News property. The retrieved content was ordered based upon the last modified time and date.

The properties above were set using XML code and they complete the query with all the necessary parameters to retrieve the relevant news contents. The contents were then displayed in a web part view which was added to default.aspx that represents the home page of *My Site* (the result is shown in Figure 24 on page 47).

5 Results and Analysis

This chapter will present our result and an evaluation of the work process and the final web part.

5.1 Result

A simple prototype of the *Corporate News Web Part* was developed and added to the user's default page on *My Site* as illustrated in Figure 24. The web part is able to get the relevant news contents by querying news items using the values stored in the user's *News* property. The web part is able to display an overview of the news contents along with the links to the news pages ordered by the modified time in the web part view.

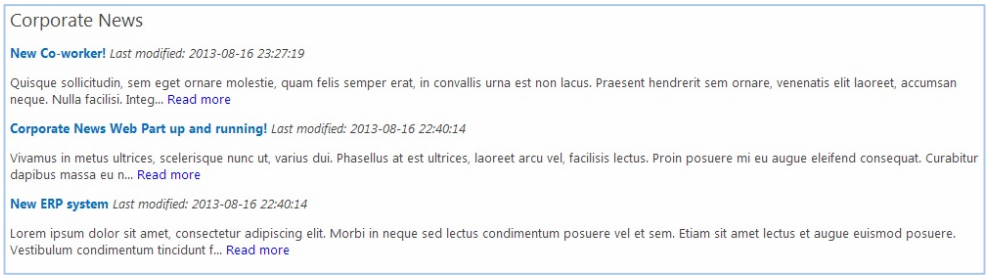

Figure 24. Corporate News web part view

5.2 Analysis

The purpose of this thesis project was to investigate if it is possible to extend the news feed in SharePoint 2013 *My Site* by adding a component that enables users to subscribe to corporate news related to different categories. The implementation met most of the goals specified in section 1.3. The result shows that it is possible to develop such a component. However, the following goal was not met: If a news article targets the user's department, then the article should be visible via the news component.

This chapter will evaluate the final web part as well as the methods and tools used during the project.

5.2.1 Tools and administration

Since the students lacked prior knowledge of SharePoint and .Net development, it was time consuming to learn how to use the tools listed in section 4.1. However, the tools worked well for the purpose of this thesis project. The students managed to learn the necessary parts of the tools, specifically how to create and to add web parts to a SharePoint site, how to configure and administer SharePoint sites and service applications, and to understand the code structure in SharePoint Visual Web Part projects.

Most of the problems the students encountered were due to the complexity of the SharePoint server. Although the solutions to most of the problems turned out to be simple, it still took some time to find these solutions. SharePoint is a large framework that consists of many components. It is not a simple product that a person can install and start using without any previous knowledge. One must invest time to study it, especially if you want to get the most out of it. Instead of initially spending time studying SharePoint, the students tried to immediately implement the web part, which turned out to be rather difficult.

As mentioned in section 4.3.1, installation of additional tools was required to be able to create SharePoint 2013 projects in Visual Studio 2012. It took a lot of time for this installation to be successful. The students followed several online tutorials without success before the final solution was found. When creating the visual web part the first time, the *sandbox solutions* approach was chosen which led to problems later during the implementation. The students encountered problems such as invalid namespaces and obsolete functions. A lot of time was wasted investigating these problems without any progress. It turned out that choosing the sandbox solution was the cause for most of these problems. As explained in section 4.3.1, many features are not supported in sandbox solution, hence the *farm solution* is recommended.

Although many problems encountered when using Visual Studio, it was still a nice tool for developing the web part. The Visual Studio environment is excellent for creating and debugging web parts [46]. Additionally, it is the primary tool for all custom code and resource development for SharePoint as it provides many features that make SharePoint development more efficient [47].

5.2.2 Project methods

Using Scrum as the project method gave a better overall project organization. Although the students did not manage to stay on track all the time, the method still helped us to see things more clearly.

As for the programming part, the students initially tried to work together. As more and more problems occurred, the work was divided into two parts: user part and company part. This way, the students could study

problems in different areas rather than both being stuck by the same problem, hence increasing productivity.

5.2.3 Corporate News Web Part

This section will evaluate the functionality and the design of the *Corporate News Web Part*.

5.2.3.1 Functionality

The fundamental functions needed for the web part are:

- A central collection of news categories should be accessible from both users and publishers.
- A user should be able to specify one or more news categories to subscribe to in their user profile of *My Site*.
- Publishers should be able to add news categories.
- The web part should be able to query news articles related to the user's news subscriptions.
- The web part should be able to display an overview of the queried news contents on the user's *My Site* default page.

The results showed that users could select multiple categories to subscribe to in the *News* property. The web part could successfully retrieve the values of the property and query for relevant news contents with them. The news publisher could tag news articles with categories and departments using the *News* and *Department* term sets in the Managed Metadata Service application. The term sets not only connect the user side and the company side, it also prohibits users and publishers from adding their own attributes to items or to user properties. They are thus both required to select values from existing term sets, which accoding to Microsoft gives more consistent use of the terms on news items that in turn leads to more relevant search results [40]. The returned news items from the query were ordered by their last modified time showing the last modified item on top. The web part could retrieve multiple news articles tagged with the same or with different categories. It could also read multiple fields of each item, such as the *Title*, *Modified*, and *Page Content* fields. The news contents were displayed in the web part view on the default page of *My Site* (along with the links to the pages for each news item). However, the web part still lacks the ability to target news to users based on their departments. The student initially thought that the implementation of this feature would be much alike the news subscription feature. The implementation turned out to be rather difficult and it required further investigation. Due to limited time, the students did not manage to retrieve the property value from the user's *Department* property in the user's profile. Hence the web part does not have

any data to work with to query the relevant news contents. Another problem that occurred at the end of the project was the query. It turned out that the web part could not specify more than two news categories in the same query. Thus if a user has subscribed to more than two categories, the web part encounters errors. I turned out that the problem was due to the XML tags in the query. Due to inexperience in XML coding, the tag <*Or*> was used to query multiple categories. However, it turned out that this tag could only allow two conditions, hence a query with a third category is not possible. Due to detecting this problem late in the development process and due to the project's limited duration, further investigation is suggested as part of future work.

Other functions that should be added to the web part include:

- The web part should be able to detect and eliminate duplications in the retrieved news items.
- The web part should save all retrieved news articles in a SharePoint list or library in the user's *My Site* site collection.

The functions above together with the realized functions should make the final web part more organized and easier to use. In short, the basic functions were realized, which enable us to show that the web part is functional and that it is possible to extend the current news feed with this additional component. However, the web part still needs a lot of improvements in order to be fully usable for the SharePoint users.

5.2.3.2 Design

The design of the web part had very low priority. The web part is not yet fully developed and the web part view only proves that the component functions. The web part has the following basic design:

- The web part displays a link title to each retrieved news article.
- The web part limits the number of the displayed news items.
- The web part limits the length of each displayed news item.

The current design is simple and easy to view. Figure 25 illustrates the design of the rendering of a single news item. However, it still needs improvements, such as:

- Displaying miniature images for news items.
- Adding a scrollbar to enable more news articles to be shown via the web part view.

- Organize the retrieved news items by dividing the view into different parts which each displays different groups of news contents.

Figure 25. A example of a retrieved news item.

5.2.4 Limitations

In summary, the *Corporate News Web Part* has the following limitations:

- The web part cannot show news content based on the user's department.
- The web part cannot make a query with more than two of the subscribed categories.
- Older news that does not fit in the web part view is not saved.
- The design is simple and does not provide picture or further organization of the displayed news contents.

There are several reasons to why the web part could not be fully implemented. Many features that worked in SharePoint 2010 are not functioning properly in 2013. When working with SharePoint 2013 projects in Visual Studio, some reference and namespaces are different from those in 2010. A lot of functions are also deprecated. As SharePoint 2013 is still new, there are few tutorials and documentations online for guidance. For these reasons the students spent a lot of time investigating problems that are new in 2013, but that would have been functional if they had used the earlier version of SharePoint. The error messages received when running the web part were unclear and didn't give any hints about where the errors were in the code (see for example the error message shown in Figure 26).

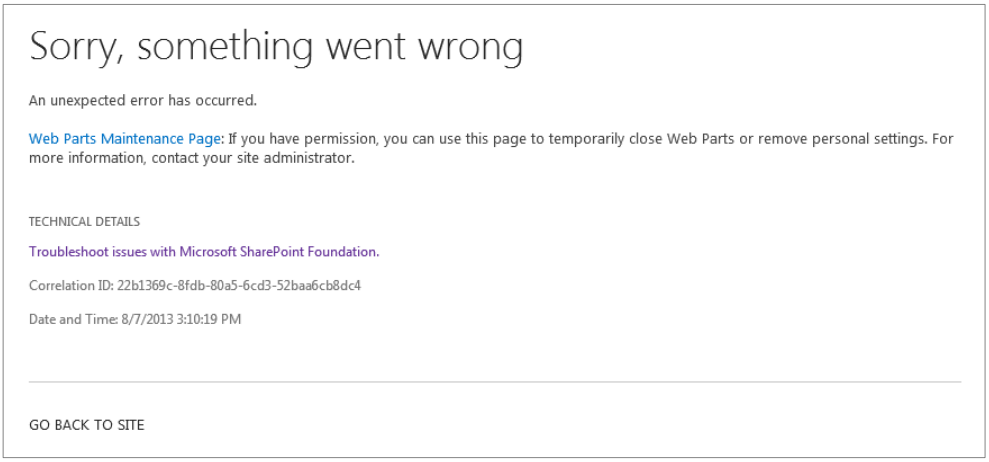

Figure 26. Error message when running the web part.

5.2.5 Other solutions

The method used for implementation of the *Corporate News Web Part* from scratch was not the only solution the students looked into. Other alternative solutions are briefly described in the following sections.

5.2.5.1 Content Query Web Part

The Content Query Web Part was presented in section 2.1.5. The web part is similar to the objective of the thesis; it aggregates and displays list items within a site collection. Although it is close to what the project is striving for, there were many features included that were unnecessary for the purpose of this thesis project. The web part is able to query different list types [48], while the implementation described in this thesis only deals with one type of list: the *Pages* list. Also, the Content Query Web Part does not automatically get the user's subscriptions, which is an important function in this thesis project. Furthermore, the web part was implemented for SharePoint 2010 and this thesis project deals with SharePoint 2013, which means a migration to the newer version of SharePoint is necessary. Many functions in SharePoint 2010 are deprecated in the later version, which leads to difficulties in migration of the web part. Rather than dealing with all these problems that may lead to much complexity, and for learning purpose, the students chose to implement the *Corporate News Web Part* from scratch which gave them knowledge in SharePoint web part development.

5.2.5.2 Proactive News Module

The Proactive News Module mentioned in section 2.1.6 has all of functionalities needed for this thesis project. The module is able to target news to relevant departments and it allows individual users to subscribe to news and updates [22]. However, the module seems to have a greater focus on the publisher side than on the user side. Unlike the *Corporate News Web Part* that focuses on users' individual subscriptions, the News Module

instead gives the publisher many choices in targeting the audience for their news items. Although the many features of this module, the students wanted to develop a web part from scratch adapting the needs of the company and to increase their own knowledge in SharePoint development.

5.2.5.3 SharePoint App

In 2012, Microsoft introduced SharePoint Apps Model as an additional feature in SharePoint 2013 to enable developers to create and add Apps to SharePoint sites [49]. Apps have their back-end code run outside of SharePoint on the host-web. These apps can be written using in many languages (such as HTML, Javascript, ASP.NET, and PHP) using a variety of development tools [50]. Unlike traditional web parts that are directly installed onto SharePoint front-end servers, Apps are deployed on SharePoint sites via an iFrame that points to the host-domain instead of the SharePoint server domain. Not only does this reduce the server load, issues such as style-sheet isolation, Javascript framework isolation, and potential cross-site scripting also disappear. Apps can be published to Microsoft's app store and can be purchased by users before installation. Overall, apps seems to be a better option than web part as it is secure, easy to develop, cloud friendly and it can do almost everything a web part can do [49]. Although there are many advantages that come with apps, the students chose the traditional web part as the final solution due to some limitations of SharePoint Apps [51]:

- Apps cannot access SharePoint server side code.
- Apps cannot access SharePoint components on other sites.
- Apps cannot include anything that is not included in a Sandboxed solution.

Since Sandboxed solutions are not preferred (as mentioned in section 4.3.1) and since the news component may need to access components from other sites in the future, the web part was chosen as the final solution.

6 Conclusions and Future work

This chapter will give a final conclusion of the thesis as well as propose future work.

6.1 Conclusions

The implemented web part meets most of the initial goals mentioned in section 1.3. It was added to the default page of the user's *My Site* and it is able to get news relevant to the user's subscription specified in the *News* property. The publisher is able to tag news articles with news categories and departments. However, the web part does not get the user's *Department* property and it can only read at most two values from *News* property at a time. Although there is room for improvements in both functionality and design, the web part proves that it is possible to extend the current *My Site* news feed with a *Corporate News Web Part*, which was the purpose of this thesis project.

The students gained both theoretical and practical knowledge in SharePoint development, as well as experience in Visual Studio environment. However, we also learned that one needs to invest time in pre-studies before developing a new part for SharePoint due to the complexity of SharePoint. With more knowledge in advance of starting the implementation, the students would have worked more efficiently when developing the web part.

6.2 Future work

This section will present the incomplete parts of the *Corporate News Web Part* and suggest future work that could both improve this part and extend it.

6.2.1 Uncompleted parts

A description of the missing functionalities is presented below in a prioritised order (based upon the desired of our employer):

- The web part should be able to retrieve more than two subscription values from the user's *News* property.

- The user's *Department* property should automatically get the user's department specified in the Active Directory.
- The web part should be able to get the user's *Department* property.
- The web part should use the retrieved data of the user's *Department* property to query for relevant news contents.

6.2.2 Suggestions for future work

Some functions that are recommended to be added to the web part in order to improve it are:

- The web part should be able to detect and delete duplicates of news items.
- The web part should be able to save old news items in a SharePoint list or library.
- The web part should be able to display a miniature image for each retrieved news item.
- The web part view should be divided into group of views. For example, one section for department news and one for the subscribed categories. Furthermore, the subscribed section could be divided into groups of categories for better organization of the news items.

Some recommendations for those that follow with regard to the implementation of the web part: invest time in studying SharePoint 2013 environment and try different things in SharePoint development in Visual Studio. Since a lot of problems with namespaces and deprecated functions occurred during our project, we learned to not strictly follow what is written in online documentations and tutorials as most of them are for SharePoint 2010 development. Many problems can be solved easily in a similar way. For example, the names of some functions have been replaced with different but similar names in the newer version. The MSDN forum [52] is a good place to look when difficulties occur.

6.3 Required reflections

Along with the growing amount of news published in corporate environments such as corporate intranets, site administrators and publishers want to organize the news contents in a non-demanding way in order to save time and money. The Corporate News Web Part provides a simple organization of news items and automatic aggregation of news contents adapted to a user's interests, making news reading easy and more time

efficient for the end users. Although this does not give a direct impact economically for the business, it increases the works efficiency of both publishers and readers. As less time is spend on finding information manually, the employers can work more efficiently creating better productivity in the long term. The web part also provides better interaction between publishers and users which gives a good social impact between them, making information sharing more easily. From the ethical part of view, the web part provides a more secure way of collecting information online. Since the users do not need to go to different pages to look for the desired content, the possibility to encounter viruses and malwares minimizes.

With the suggested improvements (as mentioned in section 6.2), the web part would enhance the social, ethical and economic benefits even more:

- The ability to target news to groups based on departments, making it even easier for publishers to spread news to the appropriate group of users.
- Even more organized news contents is desirable on the user side, in order to make it easy for users to find what they are looking for among the subscribed contents.
- Even better interaction between publishers and users is desirable since the publishers can better target their news to specific groups of users.

References

[1] D. Roe, “Report: A Third of Organizations Use SharePoint as an Enterprise CMS,” CMSWire, 12-Oct-2011. [Online]. Available: http://www.cmswire.com/cms/information-management/report-a-third-of-organizations-use-sharepoint-as-an-enterprise-cms-013004.php. [Accessed: 04-Apr-2013].

[2] FPweb.net, “Compare SharePoint 2010 and 2013 | SharePoint 2013 Cloud Hosting,” Fpweb.net. [Online]. Available: http://www.fpweb.net/sharepoint-hosting/2013/compare-sharepoint-2010-2013/. [Accessed: 04-Apr-2013].

[3] M. R. Gilbert, K. M. Shegda, G. Phifer, and J. Mann, “SharePoint 2010 Is Poised for Broader Enterprise Adoption | 1209350,” Gartner, 19-Oct-2009. [Online]. Available: http://www.gartner.com/DisplayDocument?id=1209350. [Accessed: 04-Apr-2013].

[4] Attitude Group, “What Is RSS? RSS Explained - www.WhatIsRSS.com,” Attitude Group Ltd. [Online]. Available: http://www.whatisrss.com/. [Accessed: 29-Apr-2013].

[5] WebReference, “What is RSS? (and Atom?),” WebReference. [Online]. Available: http://www.webreference.fr/defintions/rss-atom-xml. [Accessed: 29-Apr-2013].

[6] A. Green, “Official Google Reader Blog: Powering Down Google Reader,” Google Reader Blog. [Online]. Available: zotero://attachment/111/. [Accessed: 02-May-2013].

[7] B. Liu, H. Han, T. Noro, and T. Tokuda, “Personal news RSS feeds generation using existing news feeds,” in in Web Engineering, Springer, 2009, pp. 419–433.

[8] Microsoft TechNet, “About announcements list - Windows SharePoint Services - Office.com,” Microsoft Corporation. [Online]. Available: http://office.microsoft.com/en-us/windows-sharepoint-services-help/about-announcements-list-HA001161167.aspx. [Accessed: 02-May-2013].

[9] Microsoft TechNet, “Logical architecture components (SharePoint Server 2010),” Microsoft Corporation, 12-May-2012. [Online]. Available: http://technet.microsoft.com/en-us/library/cc263121.aspx. [Accessed: 02-May-2013].

[10] Amrein Engineering, “Sharepoint News Roll Up Web Part,” Amrein

Engineering AG. [Online]. Available: http://www.amrein.com/apps/page.asp?Q=5798. [Accessed: 02-May-2013].
[11] H. Tschabitscher, "NewsGator Online Services - RSS News Feed Reader Review - About Email," About.com. [Online]. Available: http://email.about.com/cs/rssfeedreaders/gr/newsgator_ols.htm. [Accessed: 02-May-2013].
[12] Microsoft TechNet, "Add RSS Feeds to your SharePoint site - SharePoint Server - Office.com," Microsoft Corporation. [Online]. Available: http://office.microsoft.com/en-us/sharepoint-server-help/add-rss-feeds-to-your-sharepoint-site-HA010291095.aspx. [Accessed: 02-May-2013].
[13] Virto Software, "Virto Social Aggregator Web Part for Microsoft SharePoint 2007 and 2010," Virto Software, 01-Aug-2012. [Online]. Available: http://www.virtosoftware.com/virto-rss-twitter-facebook-aggregator-web-part-for-sharepoint.aspx#1. [Accessed: 02-May-2013].
[14] Microsoft TechNet, "Working with SharePoint lists, Part 1 - Windows SharePoint Services - Office.com," Microsoft Corporation. [Online]. Available: http://office.microsoft.com/en-001/windows-sharepoint-services-help/working-with-sharepoint-lists-part-1-HA001119988.aspx. [Accessed: 02-May-2013].
[15] Microsoft TechNet, "Display data from multiple lists with the Content Query Web Part - SharePoint Designer - Office.com," Microsoft Corporation. [Online]. Available: http://office.microsoft.com/en-us/sharepoint-designer-help/display-data-from-multiple-lists-with-the-content-query-web-part-HA010174134.aspx. [Accessed: 02-May-2013].
[16] Microsoft TechNet, "Conceptual Overview of SharePoint Foundation," Microsoft Corporation, May-2010. [Online]. Available: http://msdn.microsoft.com/en-us/library/ee537319.aspx. [Accessed: 02-May-2013].
[17] Microsoft TechNet, "Plan for My Sites in SharePoint Server 2013," Microsoft Corporation, 15-Jan-2013. [Online]. Available: http://technet.microsoft.com/en-us/library/cc262500.aspx. [Accessed: 02-May-2013].
[18] Microsoft TechNet, "Using Central Administration (Office SharePoint Server)," Microsoft Corporation. [Online]. Available: http://technet.microsoft.com/en-us/library/cc263312(v=office.12).aspx. [Accessed: 02-May-2013].
[19] D. Holme, "SharePoint 2010: Least Privilege Service Accounts | SharePoint content from SharePoint Pro," SharePoint Pro, 14-Apr-2011. [Online]. Available: http://sharepointpromag.com/sharepoint/least-privilege-service-accounts-sharepoint-2010. [Accessed: 02-May-2013].
[20] S. Sheppard, "Overlapped Recycling And SharePoint: Why SharePoint Requires It - Steve Sheppard's Blog - Site Home - MSDN Blogs,"

Microsoft Corporation, 17-Dec-2007. [Online]. Available: http://blogs.msdn.com/b/steveshe/archive/2007/12/17/overlapped-recycling-and-sharepoint-why-sharepoint-requires-it.aspx. [Accessed: 02-May-2013].

[21] Armrein Engineering Messaging&Groupware Solutions, "SharePoint News Rollup Web Part", 2011, Copyright 2011 by Armrein Engineering Messaging&Groupware Solutions. Available: http://www.amrein.com/pic/appli/NewsRollupPictures.gif. [Accessed: 04-Apr-2013]. Appears with permission of Amrein Engineering.

[22] ProActive, 'ProActive A/S - Target different audiences on your corporate intranet with News Module for SharePoint', ProActive A/S, 19-August-2013. [Online]. Available: http://www.proactive.dk/en/Losninger/ProSolutions/NewsModule.aspx. [Accessed: 19-August-2013].

[23] Microsoft Technet, 'Chapter 2: SharePoint Architecture (Part 1 of 2)', Microsoft Corporation, 19-August-2013. [Online]. Available: http://msdn.microsoft.com/en-us/library/bb892189(v=office.12).aspx#WSS3Inside_IIS. [Accessed: 19-August-2013].

[24] Microsoft Technet, 'Chapter 2: SharePoint Architecture (Part 1 of 2)', Microsoft Corporation. [Online]. Available: http://msdn.microsoft.com/en-us/library/bb892189(v=office.12).aspx#WSS3Inside_ASP. [Accessed: 19-August-2013].

[25] Wikipedia, 'ASP.NET - Wikipedia, the free encyclopedia', Wikipedia. [Online]. Available: http://en.wikipedia.org/wiki/ASP.NET_3.5#Versions. [Accessed: 19-August-2013].

[26] Microsoft Technet, 'Chapter 2: SharePoint Architecture (Part 1 of 2)', Microsoft Corporation. [Online]. Available: http://msdn.microsoft.com/en-us/library/bb892189(v=office.12).aspx#WSS3Inside_Integration. [Accessed: 19-August-2013].

[27] Microsoft Technet, 'Logical architecture components (SharePoint Server 2010)', Microsoft Corporation. [Online]. Available: http://technet.microsoft.com/en-us/library/cc263121.aspx#section8. [Accessed: 20-August-2013].

[28] Microsoft Office, 'Introduction to sites, workspaces, and pages - Windows SharePoint Services - Office.com', Microsoft Corporation. [Online]. Available: http://office.microsoft.com/en-001/windows-sharepoint-services-help/introduction-to-sites-workspaces-and-pages-HA010021413.aspx. [Accessed: 20-August-2013].

[29] Microsoft Technet, 'Host-named site collection architecture and deployment (SharePoint 2013)', Microsoft Corporation. [Online]. Available: http://technet.microsoft.com/en-us/library/cc424952.aspx. [Accessed: 20-August-2013].

[30] Microsoft Technet, 'Logical architecture components (SharePoint Server 2010)', Microsoft Corporation. [Online]. Available: http://technet.microsoft.com/en-us/library/cc263121.aspx#section4. [Accessed: 20-August-2013].
[31] Microsoft Technet, 'Logical architecture components (SharePoint Server 2010)', Microsoft Corporation. [Online]. Available: http://technet.microsoft.com/en-us/library/cc263121.aspx#section9. [Accessed: 20-August-2013].
[32] R. Wuhrman, 'SharePoint My Site 101, slide 2', slideshare. [Online]. Available: http://www.slideshare.net/rwuhrman/sharepoint-my-site-101. [Accessed: 20-August-2013].
[33] R. Wuhrman, 'SharePoint My Site 101, slide 11', slideshare. [Online]. Available: http://www.slideshare.net/rwuhrman/sharepoint-my-site-101. [Accessed: 20-August-2013].
[34] Microsoft Office, 'Modify pages on My Site - SharePoint Server - Office.com', Microsoft Corporation. [Online]. Available: http://office.microsoft.com/en-001/sharepoint-server-help/modify-pages-on-my-site-HA001160705.aspx?CTT=5&origin=HA001160556. [Accessed: 20-August-2013].
[35] R. Wuhrman, 'SharePoint My Site 101, slide 17', slideshare. [Online]. Available: http://www.slideshare.net/rwuhrman/sharepoint-my-site-101. [Accessed: 20-August-2013].
[36] R. Wuhrman, 'SharePoint My Site 101, slide 18', slideshare. [Online]. Available: http://www.slideshare.net/rwuhrman/sharepoint-my-site-101. [Accessed: 20-August-2013].
[37] R. Wuhrman, 'SharePoint My Site 101, slide 19', slideshare. [Online]. Available: http://www.slideshare.net/rwuhrman/sharepoint-my-site-101. [Accessed: 20-August-2013].
[38] Microsoft Technet, 'Chapter 3: Pages and Design (Part 1 of 2)', Microsoft Corporation. [Online]. Available: http://msdn.microsoft.com/en-us/library/bb964680(v=office.12).aspx#WSS3Inside3_Fundamentals. [Accessed: 20-August-2013].
[39] Microsoft Technet, 'Logical architecture components (SharePoint Server 2010)', Microsoft Corporation. [Online]. Available: http://technet.microsoft.com/en-us/library/cc263121.aspx#section2. [Accessed: 20-August-2013].
[40] Microsoft Technet, 'Managing Enterprise Metadata in SharePoint Server 2010 (ECM)', Microsoft Corporation. [Online]. Available: http://msdn.microsoft.com/en-us/library/ee559337(v=office.14).aspx. [Accessed: 20-August-2013].
[41] A. Connell, 'Andrew Connell, MVP SharePoint Server - SharePoint 2010 Managed Metadata - Global vs. Local Term Sets', Andrew Connell. [Online]. Available: http://www.andrewconnell.com/blog/SP2010-Managed-Metadata-

Global-vs-Local-Term-Sets. [Accessed: 20-August-2013].
[42] Microsoft Technet, 'Changes from SharePoint 2010 to SharePoint 2013', Microsoft Corporation. [Online]. Available: http://technet.microsoft.com/en-us/library/ff607742.aspx. [Accessed: 20-August-2013].
[43] T. Quinlan, 'Setting up Visual Studio 2012 for SharePoint 2013 development offline - Tim Quinlan - MSFT - Site Home - MSDN Blogs', MSDN Blog. [Online]. Available: http://blogs.msdn.com/b/timquin/archive/2013/01/22/setting-up-visual-studio-2012-for-sharepoint-2013-development-offline.aspx. [Accessed: 20-August-2013].
[44] S. Pegg, 'SharePoint Sandbox Solutions: The Good, The Bad, and The Ugly | Pentalogic Technology', Pentalogic Technology. [Online]. Available: http://blog.pentalogic.net/2012/07/sharepoint-sandbox-solutions-the-good-the-bad-and-the-ugly/. [Accessed: 20-August-2013].
[45] Microsoft Office, 'Add and edit user profile properties - SharePoint Online for enterprises - Office.com', Microsoft Corporation. [Online]. Available: http://office.microsoft.com/en-001/office365-sharepoint-online-enterprise-help/add-and-edit-user-profile-properties-HA102772741.aspx. [Accessed: 20-August-2013].
[46] Microsoft Technet, 'A Developer's Introduction to Web Parts', Microsoft Corporation. [Online]. Available: http://msdn.microsoft.com/en-us/library/dd583154(v=office.11).aspx. [Accessed: 20-August-2013].
[47] Microsoft Technet, 'SharePoint Development in Visual Studio', Microsoft Corporation. [Online]. Available: http://msdn.microsoft.com/en-us/library/ee330921.aspx. [Accessed: 20-August-2013].
[48] Microsoft Technet, 'How to: Display Custom Fields in a SharePoint Content By Query Web Part (ECM)', Microsoft Corporation. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms497457.aspx. [Accessed: 20-August-2013].
[49] P. Katz, 'SharePoint 2013's App Model vs. SharePoint Web Parts', LimeLeap. [Online]. Available: http://go.limeleap.com/community/bid/256364/SharePoint-2013-s-App-Model-vs-SharePoint-Web-Parts. [Accessed: 20-August-2013].
[50] K. Kapoor, 'One of the big new features of SharePoint 2013 is "apps"', BrightStarr. [Online]. Available: http://www.brightstarr.com/sharepoint-2013-apps. [Accessed: 20-August-2013].
[51] N. B. Bachir, 'Apps for sharepoint 2013', SlideShare. [Online]. Available: http://www.slideshare.net/nordinebenbachir/apps-for-sharepoint-2013. [Accessed: 20-August-2013].
[52] Microsoft Technet, 'MSDN forums', Microsoft Corporation. [Online]. Available: http://social.msdn.microsoft.com/Forums/en-US/home.

[Accessed: 20-August-2013].
[53] Microsoft Technet, 'IC73000.gif (GIF Image, 610 × 235 pixels)', Microsoft Corporation. [Online]. Available: http://i.msdn.microsoft.com/dynimg/IC73000.gif. [Accessed: 20-August-2013].
[54] Microsoft Technet, 'IC165428.gif (GIF Image, 415 × 247 pixels)', Microsoft Corporation. [Online]. Available: http://i.msdn.microsoft.com/dynimg/IC165428.gif. [Accessed: 20-August-2013].
[55] Microsoft Technet, 'IC24470.gif (GIF Image, 535 × 246 pixels)', Microsoft Corporation. [Online]. Available: http://i.msdn.microsoft.com/dynimg/IC24470.gif. [Accessed: 20-August-2013].
[56] Microsoft Technet, 'IC61430.gif (GIF Image, 508 × 321 pixels)', Microsoft Corporation. [Online]. Available: http://i.msdn.microsoft.com/dynimg/IC61430.gif. [Accessed: 20-August-2013].
[57] Microsoft Technet, 'IC30429.gif (GIF Image, 875 × 262 pixels)', Microsoft Corporation. [Online]. Available: http://i.msdn.microsoft.com/dynimg/IC30429.gif. [Accessed: 20-August-2013].
[58] M. Torrisi, 'DNS 101: What Is A Hostname? | Dyn Blog', Dyn, 27-August-2012. [Online]. Available: http://dyn.com/blog/what-is-a-hostname-dns-ip-address/. [Accessed: 22-September-2013].
[59] Heise Media UK Ltd, 'Microsoft issues warning about XSS hole in SharePoint - The H Security: News and Features', The H Security, 30-April-2010. [Online]. Available: http://www.h-online.com/security/news/item/Microsoft-issues-warning-about-XSS-hole-in-SharePoint-990812.html. [Accessed: 22-October-2013].

Appendices

Appendix A: Organization

Common:

- Abstract
- Sammanfattning
- References
- Appendices

Lisan:

- Introduction
- Microsoft SharePoint Architecture & Topology

Tingting:

- Background
- Method
- Result and Analysis
- Conclusions and Future Work
- Section 3.2.7 Managed Metadata

Appendix B: Code

VisualWebPart1.ascx.cs

```
using System;
using System.ComponentModel;
using System.Text;
using System.Collections;
using System.Web.UI.WebControls.WebParts;
using Microsoft.Office.Server.UserProfiles;
using Microsoft.SharePoint;
using Microsoft.SharePoint.Client;
using Microsoft.SharePoint.Client.Search.Query;
using Microsoft.SharePoint.Utilities;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data;
using System.Xml;

namespace VisualWebPartProject5.VisualWebPart1
{
    [ToolboxItemAttribute(false)]
    public partial class VisualWebPart1 : WebPart
    {
        // Uncomment the following SecurityPermission
attribute only when doing Performance Profiling using
        // the Instrumentation method, and then remove the
SecurityPermission attribute when the code is ready
        // for production. Because the SecurityPermission
attribute bypasses the security check for callers of
        // your constructor, it's not recommended for
production purposes.
```

```
        //
[System.Security.Permissions.SecurityPermission(System.Securi
ty.Permissions.SecurityAction.Assert, UnmanagedCode = true)]

        protected override void OnInit(EventArgs e)
        {
            base.OnInit(e);
            InitializeControl();

        }

        protected void Page_Load(object sender, EventArgs e)
        {
            // Get SPSite and service context from string
            string strUrl = "http://sp2013/my";
            SPSite site = new SPSite(strUrl);
            SPServiceContext serviceContext =
SPServiceContext.GetContext(site);

            // Initialize user profile config manager object
            UserProfileManager upm = new
UserProfileManager(serviceContext);
            string username =
Microsoft.SharePoint.SPContext.Current.Web.CurrentUser.LoginN
ame;
            string sAccount = username;
            UserProfile u = upm.GetUserProfile(sAccount);

            ProfileValueCollectionBase pvc =
u.GetProfileValueCollection("News");

            string value = "";

            IEnumerator iter = pvc.GetEnumerator();

            while (iter.MoveNext())
            {
                value += (String)iter.Current + " ";
```

```
            }

            string noLastSpace = value.Substring(0,
value.Length - 1);
            string[] categories = noLastSpace.Split(' ');


            /*Get news content*/


            SPSite site2 = new
SPSite("http://sp2013/pub/how/nyheter-och-
media/corporatenews");
            SPWeb web = site2.OpenWeb();
            SPSiteDataQuery query = new SPSiteDataQuery();

            query.Lists = "<Lists ServerTemplate=\"850\" />";
//850 for page template.

            query.ViewFields = "<FieldRef Name=\"Title\"
Nullable='TRUE' Type=\"Text\"/>";
            query.ViewFields += "<FieldRef Name=\"Modified\"
Nullable='TRUE'/>";
            query.ViewFields += "<FieldRef
Name=\"PublishingPageContent\" Nullable='TRUE'/>";
            query.ViewFields += "<FieldRef Name=\"FileRef\"
Nullable='TRUE'/>";
            query.ViewFields += "<FieldRef Name=\"NewsTag\"
Nullable='TRUE'/>";
            query.ViewFields += "<FieldRef
Name=\"Department\" Nullable='TRUE'/>";


            query.Query = "<Where><Or>";

            int i = 0;
            txtBox2.Text = categories.Length.ToString();
            while(i<=categories.Length-1){
```

```
                query.Query += "<Eq><FieldRef
Name='NewsTag'/><Value Type='Text'>"+categories[i] as
string+"</Value></Eq>";
                i++;
            }

            query.Query += "</Or></Where><OrderBy>" +
                              "<FieldRef Name='Modified'/>" +
                          "</OrderBy>";

            query.Webs = "<Webs Scope=\"Recursive\" />";

            DataTable dt = web.GetSiteData(query);

            if (dt.Rows.Count != 0)
            {
                txtBox.Text = "";

                DataRowCollection rows = dt.Rows;
                int rowIndex = rows.Count-1;
                int lastIndex = rowIndex;
                while (rowIndex >= 0) {
                    DataRow row = rows[rowIndex];

                    string pubURL = row["FileRef"] as string;
                    string friendlyURL =
pubURL.Substring(pubURL.IndexOf("#") + 1);
                    string url =
"http://sp2013/"+friendlyURL;

                    string pubContent =
row["PublishingPageContent"] as string;
                    string shortContent = "";
                    if (pubContent.Length > 200)
                    {
                        shortContent =
pubContent.Substring(0, 200);
                    }
```

```
                    else
                    {
                        shortContent = pubContent;
                    }

                    txtBox.Text += "<p><a href=" + url +
"><b>" + row["Title"] as string + " </b></a>";

                    txtBox.Text += " <i>Last modified: " +
row["Modified"] as string + "</i>";

                    txtBox.Text += shortContent + "... " +
"<a href=" + url + " style='color:blue;'>Read more</a></p>";
                    if ((lastIndex - rowIndex) ==4)
                    {
                        rowIndex = -1;
                    }
                    else
                    {
                        rowIndex--;
                    }
                }
            }
            else
            {
                txtBox.Text = "Table empty.";

            }
        }
    }
}
```

VisualWebPart1.ascx

```
<%@ Assembly Name="$SharePoint.Project.AssemblyFullName$" %>
<%@ Assembly Name="Microsoft.Web.CommandUI, Version=15.0.0.0,
Culture=neutral, PublicKeyToken=71e9bce111e9429c" %>
<%@ Register Tagprefix="SharePoint"
Namespace="Microsoft.SharePoint.WebControls"
Assembly="Microsoft.SharePoint, Version=15.0.0.0,
Culture=neutral, PublicKeyToken=71e9bce111e9429c" %>
<%@ Register Tagprefix="Utilities"
Namespace="Microsoft.SharePoint.Utilities"
Assembly="Microsoft.SharePoint, Version=15.0.0.0,
Culture=neutral, PublicKeyToken=71e9bce111e9429c" %>
<%@ Register Tagprefix="asp" Namespace="System.Web.UI"
Assembly="System.Web.Extensions, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35" %>
<%@ Import Namespace="Microsoft.SharePoint" %>
<%@ Register Tagprefix="WebPartPages"
Namespace="Microsoft.SharePoint.WebPartPages"
Assembly="Microsoft.SharePoint, Version=15.0.0.0,
Culture=neutral, PublicKeyToken=71e9bce111e9429c" %>
<%@ Control Language="C#" AutoEventWireup="true"
CodeBehind="VisualWebPart1.ascx.cs"
Inherits="VisualWebPartProject5.VisualWebPart1.VisualWebPart1
" %>
<asp:Label runat="server" ID="txtBox"></asp:Label>
<asp:Label runat="server" ID="txtBox2"></asp:Label>
```

TRITA-ICT-EX-2013:207

www.kth.se