Enterprise Semantic Web

Nombre Estudiante David Gutiérrez Alba
2º ciclo de Ingeniería en Informática

Nombre Consultor Felipe Geva Urbano

Licencias alternativas (elegir alguna de las siguientes y sustituir la de la página anterior)

A) Creative Commons:

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-CompartirIgual 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-SinObraDerivada 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-CompartirIgual 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento 3.0 España de Creative Commons

B) GNU Free Documentation License (GNU FDL)

Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled "GNU Free Documentation License".

C) Copyright

FICHA DEL TRABAJO FINAL

Título del trabajo:	Enterprise Semantic Web
Nombre del autor:	David Gutiérrez Alba
Nombre del consultor:	Felipe Gerva Urbano
Fecha de entrega (mm/aaaa):	12/2012
Área del Trabajo Final:	Semantic Web
Titulación:	2º ciclo de Eng. Informática

Resumen del Trabajo (máximo 250 palabras):

Dada la creciente complejidad y competitividad, organizaciones ahogadas en una caótica masa de datos necesitan extraer algún valor de dichos datos para convertirlo en verdaderos activos de negocio. Este caos de datos es el resultado lógico de la propia naturaleza humana and merece la pena aceptar y tratar con este escenario en lugar de un entorno sistematizado y rígido que impida surgir el conocimiento. La gestión del conocimiento es actualmente un reto al que se enfrenta las organizaciones y el advenimiento de las nuevas tecnologías tales como Web semántica o novedosas plataformas colaborativas para ayudar organizaciones a recolectar, representar y distribuir conocimiento esparcido por toda la organización, nos invita a repensar la gestión del conocimiento en la organizaciones. Web semántica es una tecnología puntera diseñada para recolectar datos y representar conocimiento. Microsoft SharePoint es uno de los casos más destacados de plataformas colaborativas en las organizaciones, de gran éxito pero carente de semántica. ¿Qué pasa si Microsoft SharePoint and Web Semántica trabajarán en tándem? ¿Cómo Web Semántica puede ser convertida en Web Semántica empresarial?

Este documento es un viaje a través de los principios de la Web Semántica y Microsoft SharePoint para comprender algunas de sus ventajas y desventajas, y cómo los principios de Web Semántica pueden mimetizarse en una solución empresarial como SharePoint. Como resultado del estudio, el lector debería adquirir conocimiento de Web Semántica y Microsoft SharePoint y aprender cómo pueden ser usadas en las organizaciones.

Abstract (in English, 250 words or less):

Given the increasing complexity and competitiveness, organizations drowned in mass of chaotic data are in need of extracting some value from that data to convert them into true business assets. This chaos of data is a logical result of the own human nature and it was worth accepting this scenario to deal with it rather than systematized and rigid environments. Knowledge management is not a new challenge among organizations and the advent of new technologies such as Semantic Web or novel collaborative platforms, to help harvest, represent and distribute knowledge scattered throughout an organization, invites us to rethink knowledge management within organizations. Semantic Web is now a state-of-art technology designed to gather data and represent knowledge in the Web. Microsoft SharePoint is the one of the most foremost cases of collaborative platforms within organizations, successful in collaboration but lacking in knowledge. What if Microsoft SharePoint and Semantic Web worked in tandem? How Semantic Web can be converted into Enterprise Semantic Web?

This document is a journey through Semantic Web principles and Microsoft SharePoint in order to come to understand some advantages and disadvantages of theirs, and how Semantic Web principles can be blended into an enterprise solution like Microsoft SharePoint. As a result of such study, readership should gain insight into both Semantic Web and Microsoft SharePoint and learn how they can be used within organizations.

Keywords (between 4 and 8):
Knowledge Management, Semantic Web, SharePoint, RDF, RDFS

Index

1. Introduction 12
1.2 Project objectives 12
1.3 Approach and methodology 12
1.4 Project plan 12
1.5 Brief summary of deliverables 13
1.6 Brief description of the other chapters in the document 13
2. The current situation 15
2.1. The uphill battle for the organizations 15
2.1.1. Highly-Managed environments 15
2.1.2. Chaos and order 15
2.2. Microsoft SharePoint: A case of study 16
3. Semantic Web 19
3.1. Why do we need Semantic Web? 19
3.1.1. Anyone can say anything anytime 19
3.1.2. Putting all together 20
3.1.3. Bringing some order to the chaos 20
3.2. Semantic web Technology Stack 22
3.2.1. The representation of distributed data on the Web 23
3.2.2. Merging data from different data sources 26
3.2.3. The identity problem 28
3.2.4. Converting triples into serilizable format 29
3.2.5. Querying data 30
3.2.6. Converting Dumb data into Smart data 32
3.3. Semantic Web Application Architecture 35
3.3.1. The design of a Web Semantic Application 35
3.3.2. Adding inference to our Semantic Web Architecture 36
3.4. A brief summary of Web Semantic concepts learnt 37
4. The journey from Semantic Web to Enterprise Semantic Web 40
4.1. SharePoint Architecture 40
4.1.1. Server Farms 40
4.1.2. Web Applications 43
4.1.3. Service Applications 45
4.1.4. Web site 45
4.1.5. Site Collection 46
4.1.6. Databases 47
4.2. SharePoint Data Model 47
4.2.1. Green field development 48
4.2.1.1. SharePoint Column 48
4.2.1.2. SharePoint Content type 49
4.2.1.3. SharePoint Content Type hierarchy 50
4.2.1.4. SharePoint List and List item 51
4.2.1.5. SharePoint List View 53
4.2.1.6. Putting columns, content types and lists together 53
4.2.1.7. Lookup column and relationships between lists 56
4.2.2. Brown field development 60

4.2.2.1. Business Data Connectivity Service 60
4.2.2.2. External Content Types and External lists 61
4.2.2.3. Brown and green field development together 61
4.3. Merging data in SharePoint 62
4.4. Giving sense to data in SharePoint 70
4.4.1. Ontologies 70
4.4.2. Taxonomies and folksonomies in SharePoint 74
4.5. Data access in SharePoint 76
5. Conclusions 80
6. Bibliography 83

Figures List

- *Figure 3.1. The Map of the Internet*
- *Figure 3.2 Rice Ontology*
- *Figure 3.3 A mass of linked data with no sense*
- *Figure 3.4 A mass of linked data with sense though an ontology*
- *Figure 3.5 Semantic Web Technology Stack*
- *Figure 3.6 First approach: Data needs a common schema*
- *Figure 3.7 Second approach: Data needs to reference entities*
- *Figure 3.8 Third approach: Data needs to reference both schemas and entities*
- *Figure 3.9. Simple Triplets*
- *Figure 3.10.* Geographic data from a data source
- *Figure 3.11.* Literary data from other data source
- *Figure 3.12.* Merged data from the previous data source
- *Figure 3.13 Semantic Web Architecture*
- *Figure 4.1. SharePoint Design Sample*
- *Figure 4.2. Web Sites Contoso Sample*
- *Figure 4.3. Site Collection Contoso Sample*
- *Figure 5.1 Unrelated database tables*
- *Figure 5.2 Database tables linked by foreign key constraint (primary key)*
- *Figure 5.3 Unrelated database tables*
- *Figure 5.4 Database tables linked by foreign key constraint (primary key)*
- *Figure 5.5 Database tables linked by foreign key constraint*
- *Figure 5.6 Lookup column relationship between SharePoint lists*
- *Figure 5.7 many-to-many relationship between SharePoint lists*
- *Figure 6.1 List aggregation patterns*
- *Figure 6.2 Union List aggregation pattern*
- *Figure 6.3 Denormalized List aggregation pattern*
- *Figure 6.4 Large Lists patterns*
- *Figure 6.5 Partitioned view pattern*
- *Figure 6.6 Partitioned List with view pattern*

Tables List

- Table 2.1 SharePoint topologies
- Table 3.1. Tabular Data about Elizabethan Literature
- Table 3.2. Sample Triples
- Table 3.3. Shakespeare's Plays as qnames
- Table 3.4. Geographical data as qnames
- Table 3.5. Triples referring to URIS
- Table 3.6. Geographical data as qnames
- Table 3.7. rdf:Property for Table 3.5
- Table 3.8 Semantic Web modeling language constructions
- Table 3.9. Comparison between Relational databases and Semantic Web
- Table 3.10 Semantic Web concepts
- Table 5.1 SharePoint, Relational database and Semantic Web
- Table 5.2 Orders SharePoint List
- Table 5.3 Triples representing data stored in Orders SharePoint list
- Table 5.4 Triples representing Type information
- Table 5.5 Triples representing entire information
- Table 7.1 Products SharePoint List in Manufacturing Web Site
- Table 7.2 Products SharePoint List in partner Web site

1. Introduction

1.2 Project objectives

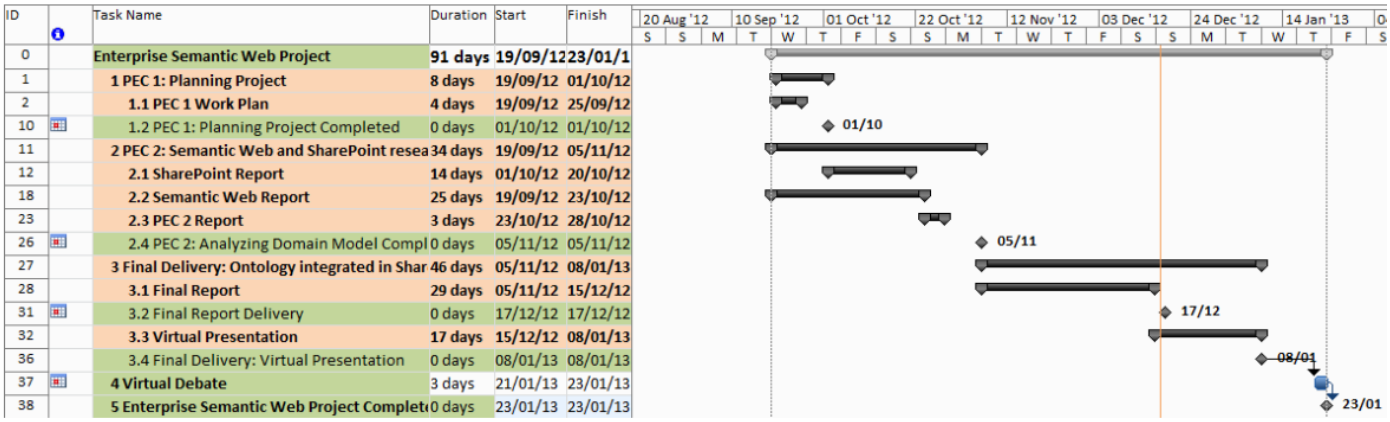
The main objectives of this project were to

- look into current challenges organizations face in regards to knowledge
- gain insight into concepts of Semantic Web
- investigate most common knowledge representation languages
- evaluate how Microsoft SharePoint 2010 deals with Semantic Web con

1.3 Approach and methodology

Due to the fact that Semantic Web finds itself in a state of art, the work entailed a deep investigation into concepts of Semantic Web along with its application to organizations. Once principles of Semantic Web were understood, the remaining work mainly had a focus on gathering SharePoint documentation and researching into how to combine Semantic Web and Microsoft SharePoint together within organizations.

1.4 Project plan



ID	Task Name	Duration	Start	Finish
0	Enterprise Semantic Web Project	91 days	19/09/12	23/01/1
1	1 PEC 1: Planning Project	8 days	19/09/12	01/10/12
2	1.1 PEC 1 Work Plan	4 days	19/09/12	25/09/12
10	1.2 PEC 1: Planning Project Completed	0 days	01/10/12	01/10/12
11	2 PEC 2: Semantic Web and SharePoint resea	34 days	19/09/12	05/11/12
12	2.1 SharePoint Report	14 days	01/10/12	20/10/12
18	2.2 Semantic Web Report	25 days	19/09/12	23/10/12
23	2.3 PEC 2 Report	3 days	23/10/12	28/10/12
26	2.4 PEC 2: Analyzing Domain Model Compl	0 days	05/11/12	05/11/12
27	3 Final Delivery: Ontology integrated in Shar	46 days	05/11/12	08/01/13
28	3.1 Final Report	29 days	05/11/12	15/12/12
31	3.2 Final Report Delivery	0 days	17/12/12	17/12/12
32	3.3 Virtual Presentation	17 days	15/12/12	08/01/13
36	3.4 Final Delivery: Virtual Presentation	0 days	08/01/13	08/01/13
37	4 Virtual Debate	3 days	21/01/13	23/01/13
38	5 Enterprise Semantic Web Project Complet	0 days	23/01/13	23/01/13

- PEC1 Work Plan is due no late than 01/10/2012
- PEC 2 Report is due no later than 05/11/2012
- Final Report is due no later than 17/12/2012
- Virtual presentation is due no later than 08/01/2013
- Debate virtual must start on 21/01/2013
- Project Completed must finish on23/01/2013

1.5 Brief summary of deliverables

The project hands in the following deliverables:

- **PEC 1 Work Plan**

PEC1 Work plan is a project deliverable which describes how the work is completed. The project scope is broken down into deliverables, work packages and activities. Addition to a project scope, the Work Plan also includes a project schedule, roles and responsibilities description as well as risk identification, risk analysis and risk response planning. The Work Plan was revised by the subject matter's consultant to check that the Work Plan meets the subject matter objectives laid down.

- **PEC 2 Report**

PEC2 Report is a simple proof of acquired knowledge of Web Semantic to complete the subsequent project activities in the project. The book "Semantic Web for the Working Ontologist, Second Edition: Effective Modelling in RDFS and OWL" by Dean Allemang and James Hendler and other documents related to Microsoft SharePoint were used as basis to catch up on Semantic Web and Microsof SharePoint..

- **Final Report**

Final Product Report is a project deliverable which summarizes the work that has been carried out and describes how the objectives have been met throughout the project. The Final Report is written by using the template provided by the subject matter's consultant.

- **Virtual Presentation**

The Virtual Presentation is a project deliverable which summarizes the work carried out and results produced throughout the project.

1.6 Brief description of the other chapters in the document

The document is broken down into four parts. The first part explains the current situation of organizations with regard to Knowledge Management. Additionally, it introduces Microsoft SharePoint as a case of study. The second part describes the main principles of Semantic Web as well as some tools and standards called Semantic Web Technology Stack. The third part spells out Microsoft SharePoint with a focus on Semantic Web. The third part is also divided into different sections. Firstly, SharePoint data model is described as well as SharePoint integration with both local and external systems. Secondly, the process of merging data from distinct data sources and giving sense to data is explained in SharePoint. Finally, the way of how SharePoint accesses to data is outlined. The document concludes with some conclusions on both SharePoint and Semantic Web.

PART I

"If only HP knew what it knows it would make three times more profit tomorrow" - Lew Platt, ex CEO Hewlett Packard

2. The current situation

Managing an organization, getting services from it as a client or collaborating with it as partner is nowadays much more complex and involves handling a great deal of information than it was in the past. To cope with this complexity, an organization depends more and more on their capacities to identify, create, represent, and distribute knowledge among organization members.

2.1. The uphill battle for the organizations

Currently, not only does an organization face technological challenges but it also has to change its mentality about how to manage itself. The management's challenge may be far harder than the technological one and it can become an uphill battle for the organization. We shall have a look at some of these changes which an organization has to be carried out in order to identify, create, represent, and distribute knowledge.

2.1.1. Highly-Managed environments

Managers in the organization and allied efforts often allow their thinking to be shaped by highly structured and tightly governed management derived entirely from a production environment. Such organizations usually think of people as parts of the big organizational machine that must blindly follow the processes, policies and procedures that are defined in order to share out their knowledge.

However, to manage thinking workers effectively and a highly collaborative environment from which the knowledge of organizations can surface, we need to foster an atmosphere in which processes are not systematized by imposing rigid processes. Fostering an atmosphere that doesn't allow workers to have a say simply makes people defensive and the team sociology can suffer grievously. However, an organization in which workers always have a say in something and are allowed to raise their voices is a thinking organization – that is, capable of creating, representing and distribute knowledge.

2.1.2. Chaos and order

Following our discourse on highly-managed environment, we can observe the manager's anxiety for having everything under control. Some managers are incapable of accepting the crude reality: they are working with people whose individual views and set of data are chaotic due to the fact that data itself is created by people. However, this chaotic data can become valuable if right relationships are established.

Therefore, it appears to be reasonable that an organization should acquire consciousness of the real nature of data and the origin of data. The organizational effort should be also aimed at dealing with chaotic data

somewhat which allows the organization to put data in order and provide them some valuable semantic sense – that is, shaping and modeling an ordered knowledge from misleading chaotic data.

2.2. Microsoft SharePoint: A case of study

In their eagerness to govern knowledge, organizations have tried out different collaborative software solutions. Microsoft SharePoint has become a successful platform whose basis is mainly collaboration. SharePoint provides a valuable enterprise solution which allows organizations to host multiple Web sites, in which organization members can collaborate by sharing documents, publishing reports to help make better decisions... So far, So good! However, if we delve into SharePoint, it raises some issues which are worthy to be considered.

The uncontrollable proliferation of untrustworthy data

Although SharePoint ships with excellent collaborative functionalities, when SharePoint is used in large- and medium-size organizations, the reality will quickly scale back to "a simply place to store documents", or to make things worse, it will soon degenerate to, "a place to lose documents". It is also increasingly common for end users (the poor worker operating the Web Browser) to find inconsistent data, data out of synchronization, and simply data disconnected from the rest of the organization. Both the enormous mass of data and its lack of semantic seem to be the root cause of this problem. Before we come to conclusions, we shall gain a bit more insight into the problem by deepening a bit into typical SharePoint deployments in numbers to come to understand its complexity in its entirety. Table 2.1 shows different kind of topologies from small topologies to large topologies in SharePoint along with the volume of items and users it can support.

Table 2.1 SharePoint topologies

Type of topology		Volume of items	Number of users
Limited deployments	One-tier farm	0-1M	< 100 users
Limited deployments	Two-tier farm	0-1M	Up to 10,000 users
Small farm deployment	Two-tier small farm	1-10M	10,000-20,000 users
Small farm deployment	Three-tier small farm	10 -20M	10,000-20,000 users
Medium farm deployment	Three-tier medium farm	20-40M	*Typically up to 50,000 users
Large farm deployment	Topologies with server groups	40 -100M	*Typically up to 50,000 users

* The factor is 10,000 users per Web server deployed.

From the data shown above, we can bring some examples that give us some clues on the nature of the problems that can occur in a typical SharePoint deployment within an organization.

- **Inconsistent data.** A worker consults a the public Human Resources Web site, seeking for the template document about how to get your tickets restaurant, and he finds a formidable template but it makes reference to another template which belongs to the Accountability department so that you can account for every Euro that you spend. So he clicks on the Web Site link and search for the template. Although the template didn't appear in the search results since it does not seem to be a very popular document, he can find the template in the end but, to his surprise, he finds out that the template can be applied to account for any expenses except for tickets restaurant. What is going on here? His boss reminds him that tickets restaurant will be included in the fringe benefits in the next days. Does it mean that the tickets restaurant doesn't exist yet? he wonders.

- **Data disconnected**. A project manager is planning to start off a new important project in the organization and he needs to get the right people with the appropriate skills and availability. The Human Resources Web Site lists the resources, however, no sign of their skills or availability. Then, he goes to the Project Center Web Site with the hope of finding the information he needs, when all of a sudden, he finds out that there is information about people's skills but not about their availability. As his patience is getting thinner, he gives up searching for further information and decides to get down to working in the project. No time to waste. As soon as the project kicks off, he will learn that the most of resources he had booked will be already committed. He wonders why nobody simply got all the information together.

- **Data out of synchronization.** A CEO in a European branch is looking into about the organization's objectives and principles about the organization, and he finds a comprehensive Web page in the Central Web Site in USA about the organization objectives this year. However, when he accesses to his branch's Web Site in his regional site, to his surprise, he finds out that his branch in Europe have different objectives to the company, what's more, it is likely that a great many people had been working on many useless projects. It is now when he realizes because there had been so many misunderstanding between managers and workers in the last few months. He wonders why the organization didn't simply update the Web Page. They would have saved the branch from a great deal of useless work.

In conclusion, even though SharePoint ships with powerful search and collaborative tools to deal with mass of data, it still remains the familiar limitation of today's Web pages - a proliferation of untrustworthy content with nonsense.

PART II

“Have you tried to turn it off and turn it on, again?”
Roy Trenneman (The IT crowd)

3. Semantic Web

So far, we have analyzed the problem and we have observed that despite having a good collaborative solution such as SharePoint, it does not guarantee that we can identify, create, represent, and distribute knowledge. It is now the time to introduce Semantic Web as a possible solution to the question given.

3.1. Why do we need Semantic Web?

3.1.1. Anyone can say anything anytime

The essential notion of the Web is the idea of an open community: Anyone can say anything anytime (AAA Slogan). This openness has resulted in massive load of Web pages that covers comprehensively topics, so to speak, almost everything you can address in your Web Browser is a Web Page. Surfing the Web for information is sometimes a dreadful nightmare. There are billions and billions of Web pages, links, labels and so on and so forth. Typically, anywhere offers us information on a topic but nowhere lets us delve sufficiently into it – that is, the Internet is wide but not deep. Sometimes, information is misleading and even contradictory at times –that is, data is often inconsistent and data sources are not integrated. To sum up, we get the impression that we get lost within such an enormous mass of nonsense information and even if we find such information, it is unlikely to be trustworthy. The question is how to build a more integrated, consistent and deep Web experience with reliable information? The Map of the Internet is shown is the Figure 3.1 below in which we can observe its complexity in its entirety.

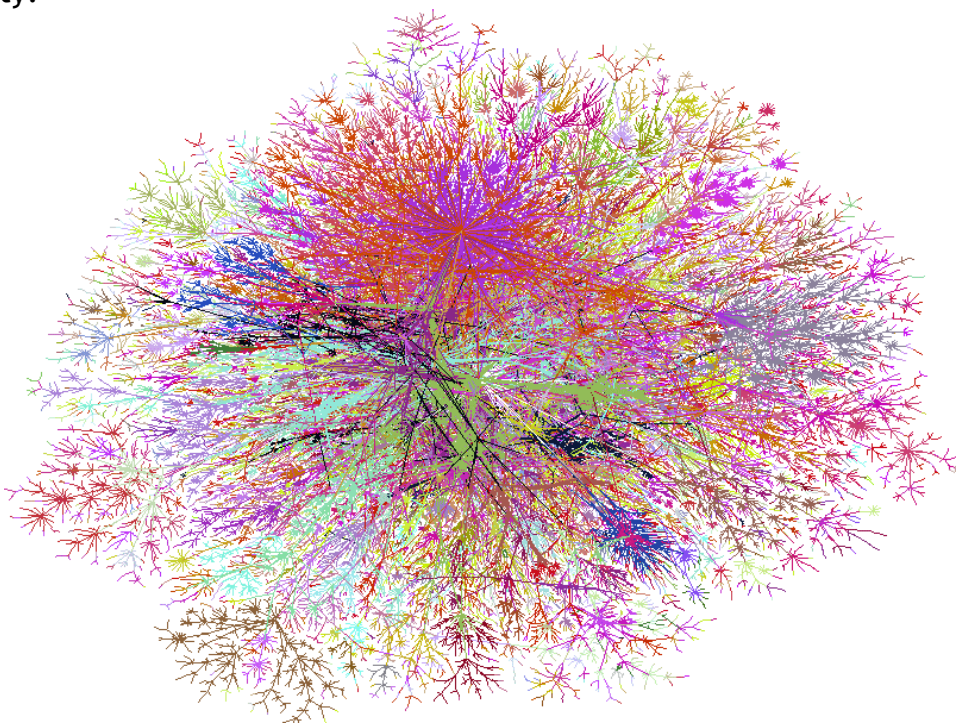


Figure 3.1. The Map of the Internet

3.1.2. Putting all together

The foremost experts of the Web agree that if there are billions and billions of unmanageable information out there, it makes sense to begin to wonder why we do not make an effort to put all this information together in a more manageable way. If so, we need a sort of infrastructure on which data can be stored, queried, indexed and crawled. But what it is clear is that something must be done with more structured data and novel query engines. The following questions can raise when we get down to coming up with a solution to this problem.

- How do we find the right file?
- How do we integrate the data?
- How do we know all those files belong there?
- How do we keep up with all these data sources?
- How do we filter data to create value?
- How do we avoid filter's filter's filter's filter's filter's data?
- How do we give sense to all this mass of data?
- How do we handle the wilderness?

The answers to these questions outlined above are the challenge for the Semantic Web. However, as we can gather from this vision, Semantic Web does not fit within the idea of making smarter Web Applications a smarter Semantic Web Technology infrastructure for integrating information, providing the consistency and availability of Web data.

3.1.3. Bringing some order to the chaos

With a Semantic Web infrastructure combining data from multiple data sources and humans constantly messing around out there, the next challenge is about how to bring any order to the chaos after having put all data together.

The ontology provides a way to make date sensible from distributed web of data by using the Semantic Web modeling languages. According to the definition in the Wikipedia, a ontology "is the philosophical study of the nature of being, existence, or reality, as well as the basic categories of being and their relations.". Figure 3.2 below shows a typical example of the rice ontology.

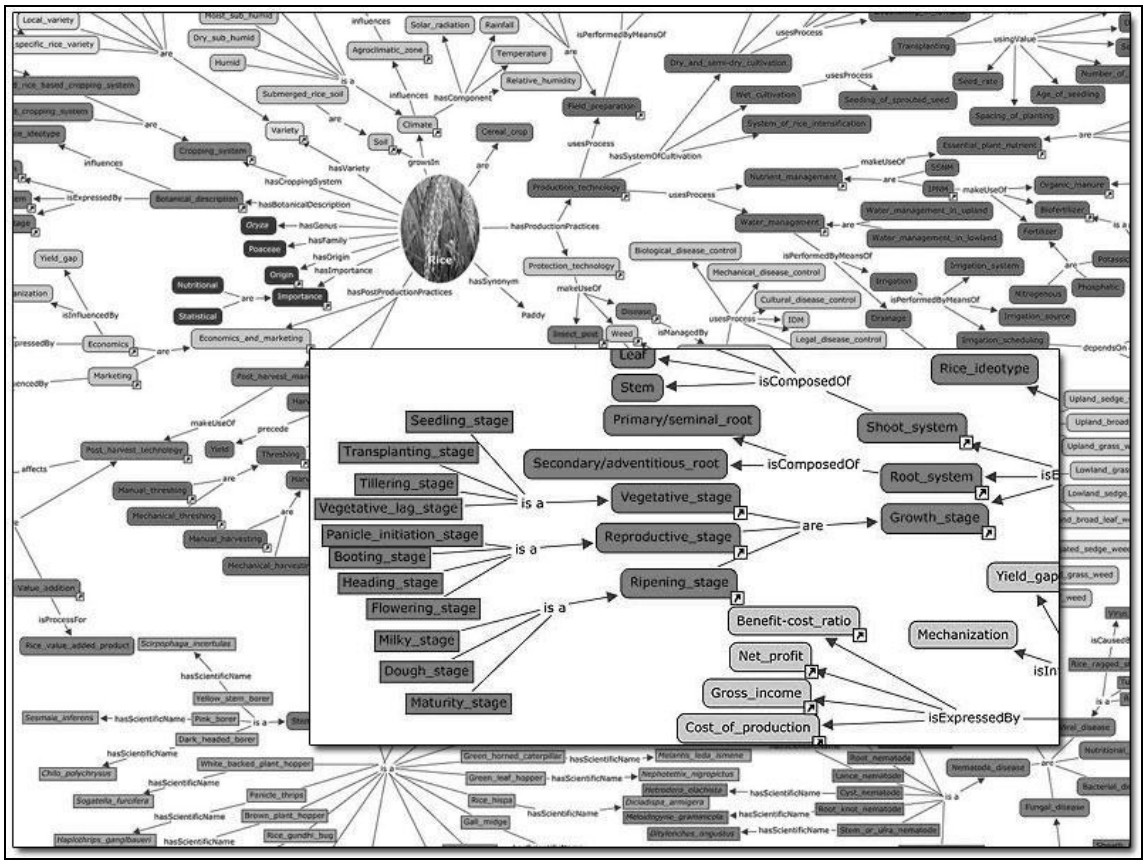


Figure 3.2 Rice Ontology

The ontology is the way we can represent knowledge within a domain in the wilderness by defining a set of concepts and relationships between them. Besides this, we also need to come up with a way to provide common vocabulary and taxonomy that allows people to have a common understanding on a domain. Figure 3.3 and Figure 3.4 below show an example of transformation of a mass of linked data into ontology in which concepts and their relationships provide sense.

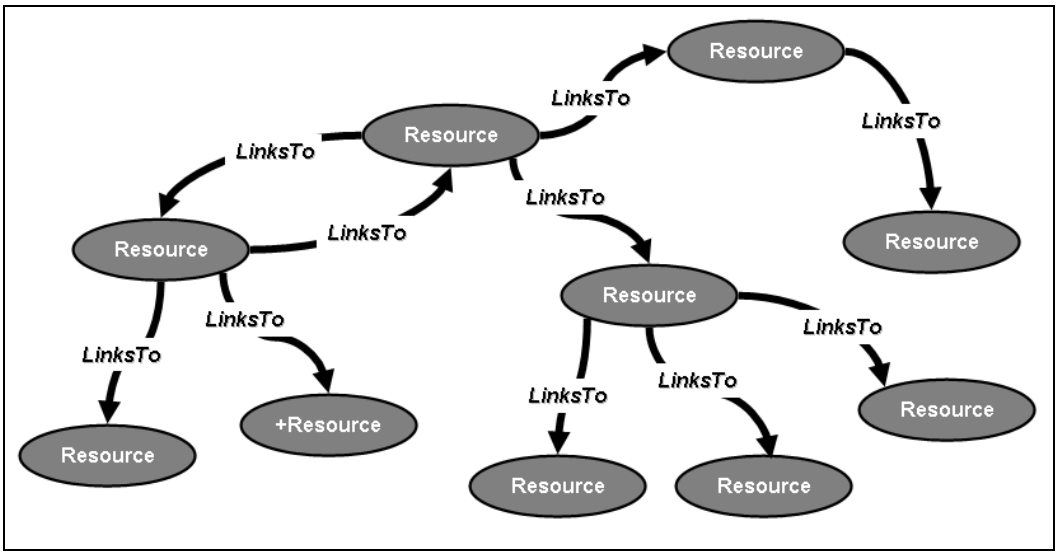


Figure 3.3 A mass of linked data with no sense

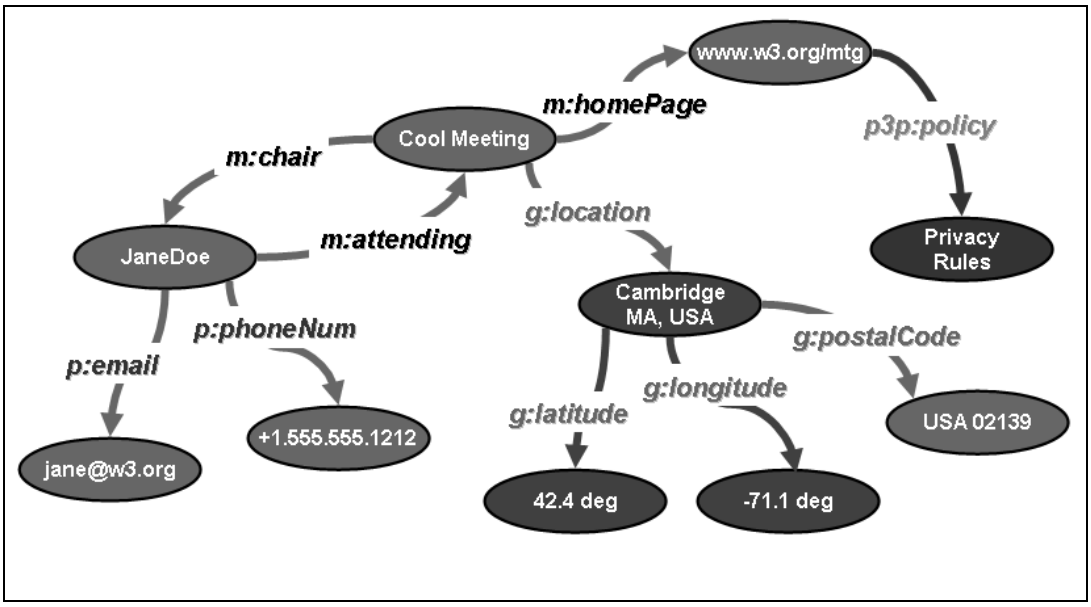


Figure 3.4 A mass of linked data with sense though an ontology

3.2. Semantic web Technology Stack

The Semantic Web Technology Stack is set of standards and protocols used to create a smarter Semantic Web infrastructure. The mission of Semantic Web Technology Stack is to provide necessary tools to putting together all data from different data sources and give some order the chaos of data which just needs to get the right data to the right place so the Web Applications can do their work. The Semantic Web Technology Stack is shown and described in the Figure 3.5

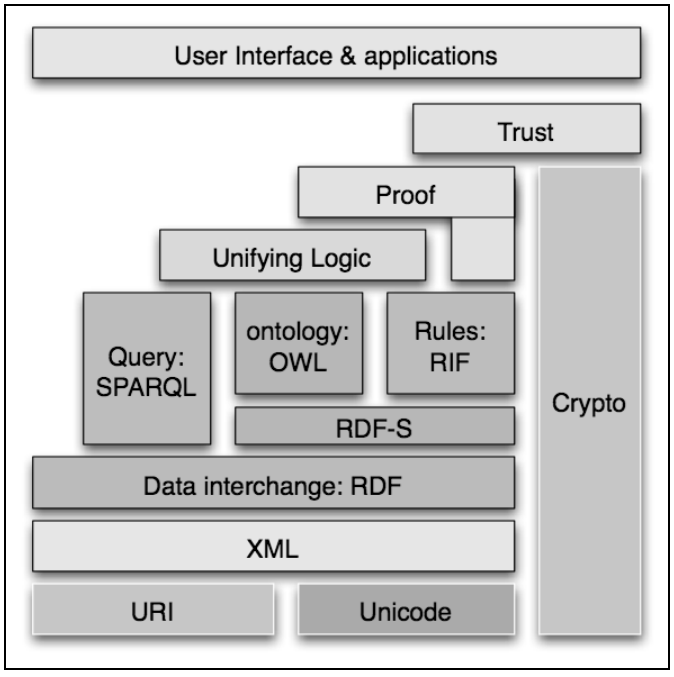


Figure 3.5 Semantic Web Technology Stack

- **Assigning unambiguous names (URI)**
- **Syntax (Turtle, N-Triples, XML/RDF)**
- **Expressing data, including metadata (RDF and RDFS)**
- **Capturing ontologies (OWL)**
- **Query (SPARQL)**
- **Rules (RIF, Rule Interchange Format)**
- **Deployment, application spaces, logic, proofs, trust**

In next sections, we will give you a ride along the concepts of Semantic Web while we introduce Semantic Web Technology Stack standards and protocols. We do not intend to give you a comprehensive view of Semantic Web but only those concepts of Semantic Web on which SharePoint can have a focus. A comprehensive explanation would be far too heavy and beyond this survey.

3.2.1. The representation of distributed data on the Web

In relational database, data are represented in tabular form. Each row represents some item, each column refers to some property we are describing and cells hold the particular values. Table 3.1 shows a tabular data representation about Elizabethan Literature and Music.

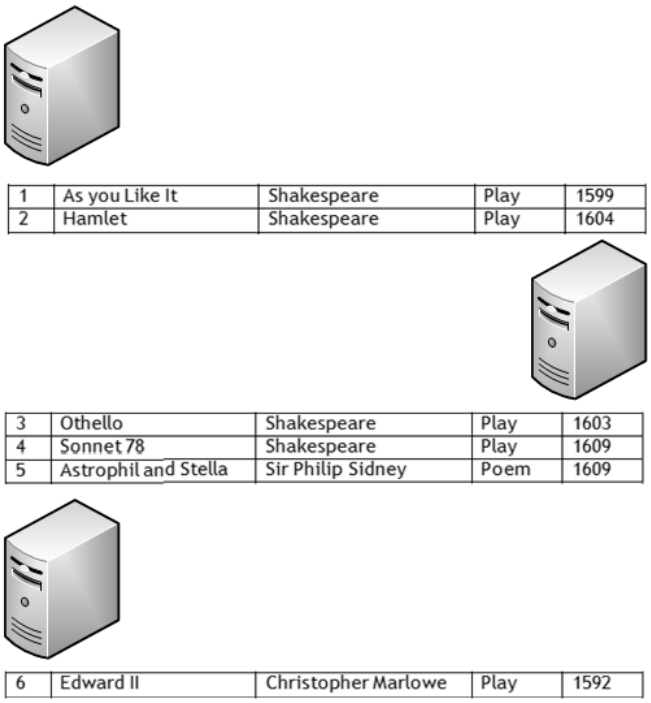
Table 3.1. Tabular Data about Elizabethan Literature

ID	Title	Author	Medium	Year
1	As you Like It	Shakespeare	Play	1599
2	Hamlet	Shakespeare	Play	1604
3	Othello	Shakespeare	Play	1603
4	Sonnet 78	Shakespeare	Play	1609
5	Astrophil and Stella	Sir Philip Sidney	Poem	1609
6	Edward II	Christopher Marlowe	Play	1592
7	Hero and Leander	Christopher Marlowe	Play	1592

In Distributed Systems, there are a few different approaches to hold this data over the Web. All these approaches have in common that some part of the data can be held on one server, whereas other servers hold the other data. Server is responsible for maintaining the information about one or more complete rows from the table and answer any query about an entity corresponding to a set of rows.

First approach: Data needs a common schema

In the First Approach to distributing data over many servers, each server is responsible for maintaining the information about one or more complete rows from the table. Any query about an entity can be answered by the computer that stores its corresponding row. The First Approach is shown in the Figure 3.6 below.



1	As you Like It	Shakespeare	Play	1599
2	Hamlet	Shakespeare	Play	1604

3	Othello	Shakespeare	Play	1603
4	Sonnet 78	Shakespeare	Play	1609
5	Astrophil and Stella	Sir Philip Sidney	Poem	1609

6	Edward II	Christopher Marlowe	Play	1592

Figure 3.6 First approach: Data needs a common schema

Despite the scability this approach can provide, each server must share information about the columns. Does the second column on one server correspond to the same information as the second column on another server? – In other words, it is required an agreed-on common schema that defines which property each column corresponds to.

Second approach: Data needs to reference entities

In the Second Approach to distributing data over many servers, each server is responsible for one or more complete columns from the table. This solution is flexible in a different way to the previous approach in that each server can be responsible for one or more kinds of information. In case there is some type of data that is not often queried, we could move to a low-performance server. The Second Approach is shown in the Figure 3.7 below.

ID	Title
1	As you Like It
2	Hamlet
3	Othello
4	Sonnet 78
5	Astrophil and Stella
6	Edward II
7	Hero and Leander

Medium	Year
Play	1599
Play	1604
Play	1603
Play	1609
Poem	1609
Play	1592
Play	1592

Author
Shakespeare
Shakespeare
Shakespeare
Shakespeare
Sir Philip Sidney
Christopher Marlowe
Christopher Marlowe

Figure 3.7 Second approach: Data needs to reference entities

This approach is similar to the previous one described in that it requires some coordination between the servers. In this case, the coordination has to do with the identities of the entities to be described. How do I know that row 3 on one server refers to the same entity as row 3 on another server? – In other words, this approach requires a global identifier for the entities.

Third approach: Data needs to reference both schemas and entities

In the Second Approach to distributing data over many servers, there is a combination of the previous approaches, in which information is neither distributed by row nor by column but is instead distributed by cell. Each server is responsible for some number of cells from the table. The Third Approach is shown in the Figure 3.8 below.

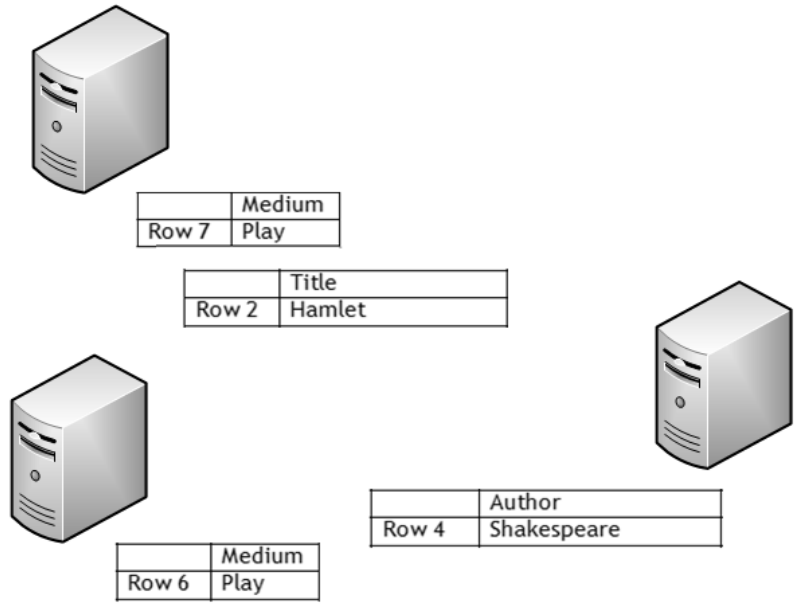


Figure 3.8 Third approach: Data needs to reference both schemas and entities

In addition to the strengths of the other previous strategies, this approach also combines the costs of the other two previous approaches. Not only do we now need a global reference for the columns, but we also need a global reference for the rows. In fact, each cell has to be represented with three values: a global reference for the row, a global reference for the column, and the value in the cell. This representation of data is called triple and the basic building block for RDF. This is exactly the sort of flexibility we look for if we want our data distribution system to really support "Anyone can say Anything Anytime". Triple is the basic building block for RDF (Resource Description Framework)

- The identifier for the row is called the subject of the triple
- The identifier for the column is called the predicate of the triple
- The value in the cell is called the object of the triple.

When more than one triple refers to the same thing, sometimes it is convenient to view the triples as a directed graph as shown in Figure 3.9. The graph visualization in Figure 3.9 expresses the same information presented in Table 3.2. However, everything we know about Shakespeare is displayed at a single node. The process to bring back all data together from different data sources is known as merge process that will be described soon.

Table 3.2. Sample Triples

Subject	Predicate	Object
Shakespeare	Wrote	King Lear
Shakespeare	Wrote	Macbeth
Anne Hathaway	Married	Shakespeare
Shakespeare	Lived In	Stratford
Stratford	Is in	England
Macbeth	Set in	Scotland
England	Part of	The UK
Scotland	Part of	The UK

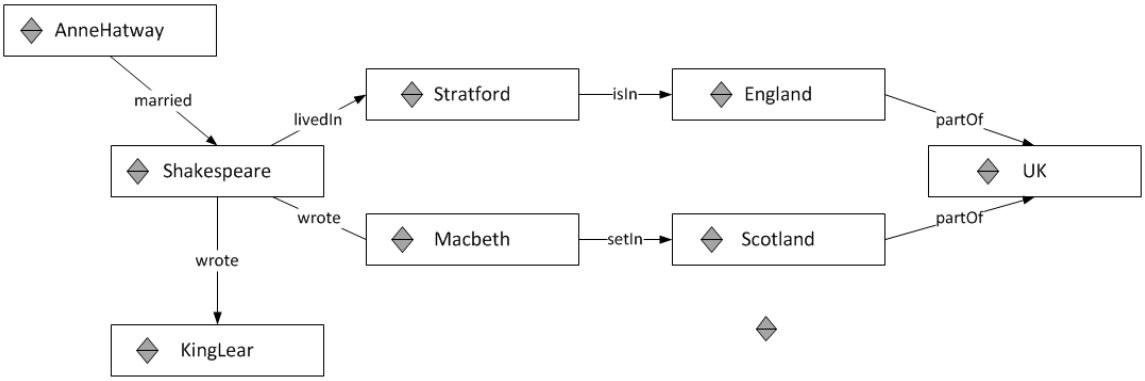


Figure 3.9. Simple Triplets

3.2.2. Merging data from different data sources

So far, we have outlined different approaches to distribute data over several data sources in the Web and we have introduced RDF as the standard to describe distributed data, the triples. However, when we want to use that data, it is pretty common that we come across those data spread out in multiple data sources and we want all them together. Merging information from two graphs is as simple as gathering the graph of all of the triples from each individual graph into a single graph. Figure 3.10 below shows geographic data from a data source whereas Figure 3.11 shows literary data from other different data source. Finally, Figure 3.12 shows merged graph from the two previous data sources.

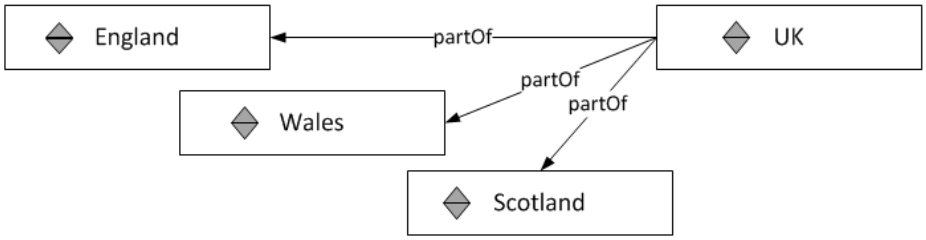


Figure 3.10. Geographic data from a data source

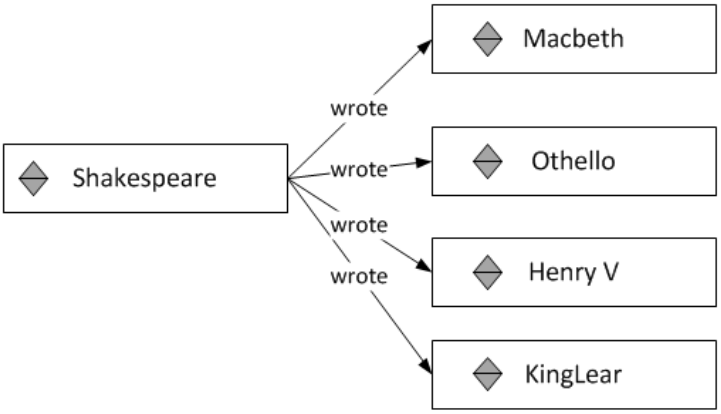


Figure 3.11. Literary data from other data source

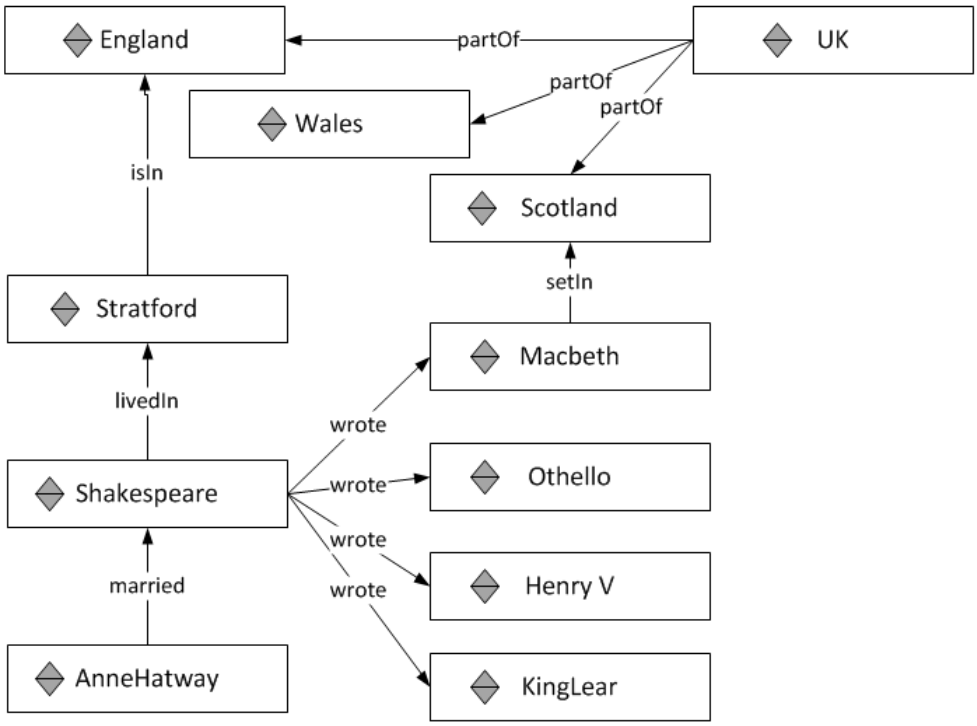


Figure 3.12. Merged data from the previous data source

3.2.3. The identity problem

The process of merge raises the following question: "What if a node in one graph was the same node as a node in another graph?" This issue is resolved through the use of Uniform Resource Identifiers (URIS). A URI provides a global identification for a resource in the Web. If any two agents in the Web want to refer to the same resource, they must agree to a common URI for such resources in advance.

RDF applies the notion of the URI to resolve the identity problem in the process of merging. A node from one graph is merged with a node from another graph, only if they have the same URI. As a matter of fact, we use qnames rather than URI in RDF. qnames are URI abbreviation which is composed of two parts: a namespace and an identifier, written with a colon between. For example, the qname representation for the identifier England in the namespace geo is simply geo: England rather than http://www.geo.com/England#. Table 3.3 and 3.4 show triples as qnames.

Table 3.3. Shakespeare's Plays as qnames

Subject	Predicate	Object
li:Shakespeare	li:wrote	li:AsYouLikeIt
li:Shakespeare	li:wrote	li:HenryV
li:Shakespeare	li:wrote	li:LovesLaboursLost
li:Shakespeare	li:wrote	li:MeasureForMeasures
li:Shakespeare	li:wrote	li:WinterTale
li:Shakespeare	li:wrote	li:Hamlet
li:Shakespeare	li:wrote	li:Othello

Table 3.4. Geographical data as qnames

Subject	Predicate	Object
geo:Scotland	geo:partOf	geo:UK
geo:England	geo:partOf	geo:UK
geo:Wales	geo:partOf	geo:UK
geo:IsleOfMan	geo:partOf	geo:UK
geo:Scotland	geo:partOf	geo:UK

Table 3.5. Triples referring to URIS

Subject	Predicate	Object
li:Shakespeare	li:wrote	li:KingLear
li:Shakespeare	li:wrote	li:Macbeth
bio:AnneHateway	bio:married	li:Shakespeare
bio:AnneHateway	bio:livedIn	li:Shakespeare
geo:Stratford	geo:isIn	Geo:England
geo:Scotland	geo:partOf	geo:UK
geo:England	geo:partOf	geo:UK

The RDF standard itself even takes advantage of the power of namespace and qnames to define its keywords in a namespace defined in the own standard, a namespace called rdf. In this way, we have rdf:type is a property that provides an elementary typing system in RDF or rdf:property to indicate when

other identifier is to be used as a predicate. Table 3.6 and 3.7 show data defined through rdf:type and rdf:property.

Table 3.6. Geographical data as qnames		
Subject	**Predicate**	**Object**
lit:Shakespeare	rdf:type	lit:Playwright
lit:Marlowe	rdf:type	lit:Playwright

Table 3.7. rdf:Property for Table 3.5		
Subject	**Predicate**	**Object**
lit:Wrote	rdf:type	rdf:Property
lit:SetIn	rdf:type	rdf:Property
bio:Married	rdf:type	rdf:Property
bio:LivedIn	rdf:type	rdf:Property
geo:IsIn	rdf:type	rdf:Property
geo:PartOf	rdf:type	rdf:Property

3.2.4. Converting triples into serilizable format

So far, so good! We already know how to represent distributed data from multiple sources and how to merge them all into a single data source to be handled, but we also need a format that can be processed by computers. Would you not expect us to bring all those data together by hand? There are some processable-computer formats available to us.

- **N-Triples.** N-Triples is a line-based, plain text serialization format for RDF graphs and corresponds directly to the raw RDF triples. It refers to resources using their fully unabbreviated URIs (no qnames).

```
<http://www.ElizabethanLiterature.org/Playwrights.rdf#Playwright>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.Geography.org/Europe/UK.rdf#Stanford>
```

- **RDF/XML** is a syntax defined by the W3C to express an RDF graph as an XML document. According to the W3C, "RDF/XML is the normative syntax for writing RDF".RDF (Resource Description Framework) is the data model used to represent for semantic web resources. RDF/XML is seen by some as the machine readable form of RDF with Notation 3 as a more human-readable form.

```
<rdf:RDF
xmlns:lit"http://www.ElizabethLiterature.org/Playwrights.rdf#"
xmlns:rdf"http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<lit:Playright>
```

```
      <lit:Id>1</lit:Id>
      <lit:FirstName>William</lit:FirstName >
      <lit:LastName>Shakespeare</lit:LastName >
      <lit:BirthPlace>Stratford</lit:BirthPlace >
      <lit:BirthDate>1564</lit:BirthDate >
</lit:Playright >
</rdf:RDF>
```

- **Turtle (Terse RDF Triple Language)** is a serialization format for the Resource Description Framework (RDF) data model. Turtle does not rely on XML and is generally recognized as being more readable and easier to edit than its XML counterpart. SPARQL (we will see it soon), the query language for RDF, uses a syntax similar to Turtle for expressing query patterns, hence it will be the serilizable format we will use in the remainder of the survey.

```
@prefix lit:
<http://www.ElizabethanLiterature.org/Playwrights.rdf#>
@prefix rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#

lit:Shakespeare rdf:type lit:Playright;
lit:Playright_ID "1";
lit:Playright_FirstName "William";
lit:Playright_LastName "Shakespeare";
lit:Playright_BirthPlace "Stratford";
lit:Playright_BirthDate "1564";
```

3.2.5. Querying data

A serilizable representation of data is useless without some means of accessing that data. The standard way of accessing RDF data is a Query Language called SPARQL (SPARQL Protocol And RDF Query Language). The SPARQL query language, which shares many similar features with other query languages such as SQL, is always the way to query data in RDF graph. SPARQL uses syntax similar to Turtle for expressing query patterns. In addition to SPARQL, there is an additional extension to the SPARQL, SPARUL, or SPARQL/Update that provides the ability to insert, delete and update RDF data held a data repository.

In order to give an idea of how SPARQL query language works, we will show some simple examples but we will not deepen into it since it is besides the objective of the survey.

RDF Data serialized in Turtle to be queried by SPARQL

```
lit:Shakespeare lit:wrote lit:KingLear
lit:Shakespeare lit:wrote lit:MacBeth
bio:AnneHathaway bio:married lit:Shakespeare
bio:AnneHathaway bio:livedWith lit:Shakespeare
lit:Shakespeare bio:livedIn geo:Stratford
geo:Stratford geo:isIn geo:England
geo:England geo:partOf geo:UK
geo:Scotland geo:partOf geo:UK
geo:Ireland geo:partOf geo:UK
lit:MacBeth lit:setin geo:Scotland
lit:KingLear lit:setin geo:England
```

Where did Shakespeare live?

```
SELECT ?place .
WHERE { lit:Shakespeare bio:livedIn ?place  .}
```

?place
geo:Stratford

What playwrights were written by Shakespeare?

```
SELECT ?playwright .
WHERE { lit:Shakespeare lit:wrote ?playwright  .}
```

?playwright
lit:KingLear
lit:MacBeth

What playwrights written by Shakespeare were set in Scotland?

```
SELECT ?playwright .
WHERE { lit:Shakespeare lit:wrote ?playwright  .
        ?playwright lit:setIn geo:Scotland  .}
```

?playwright
lit:MacBeth

3.2.6. Converting Dumb data into Smart data

Just at the beginning of this survey, in the section "The uncontrollable proliferation of untrustworthy data", we introduced the problem of disconnected, inconsistent and desynchronized data. In the world of Semantic Web, sometimes this situation is known as "dumb" data and the main objective of Semantic Web is to provide a more connected Web infrastructure. RDF ships with a consistent way to represent data so that information from multiple sources can be brought together and merge into a single data source to be queried by SPARQL. The problem arises when we want to use that data, it is just when the differences between data surface and we need some sort of mechanisms to deal with them.

"Dumb data" example:

Imagine we are in the webpage of Elizabethan Literature organization, and we search for "Shakespeare" in the category of "Authors." Our search comes up empty. We are surprised, because we were quite certain that we know that Shakespeare is a "Playwright" and, therefore, an "Author". So we look up the name "Shakespeare" but on this occasion we search "Shakespeare" in the category "Playwrights". There exists "Shakespeare" as a "Playwright. What is going on? This situation is that Semantic Web would define as "dumb data". If "Shakespeare" is a "Playwright" and "Playwright" is a subcategory of "Author". Wouldn't it be logic that "Shakespeare" shows in the search results for both categories? What would we convert this "dumb data" into "smarter data"? How can we express this meaning in a way that is consistent and maintainable?

To make our data look more connected and consistently integrated, we must be able to make date sensible from distributed web of data by adding relationships into the data. In this example, we want to be able to express the relationship between "Author" and "Playwright" that will tell us that any item in the "Playwright" category should also be in the "Author" category.

The Semantic Web approach to this problem uses a modeling language in which the relationship between the sources can be described and the meaning of the modeling language. This is known as ontology. Meaning of data is given by patterns of inference and data integration is achieved by invoking such inferences; a query returns not only the asserted data but also inferred data. The Semantic Web approach also uses taxonomies to provide controlled vocabularies to guarantee a common background of terms. Just as do ontologies, taxonomies play an important role since it allows us to define broad and narrow terms and bring together different name to the same thing.
The Semantic Web Modeling Languages differ mainly in their level of expressivity and are:

- **RDF (Resource Description Framework)**. This is the basic framework that the rest of the Semantic Web is based on. RDF provides a mechanism for allowing anyone to make a basic statement about anything.

- **RDFS (RDF Schema language)**. RDFS is a language with the expressivity to describe the basic notions of commonality and variability familiar from oriented-object languages — namely classes, subclasses, and properties.

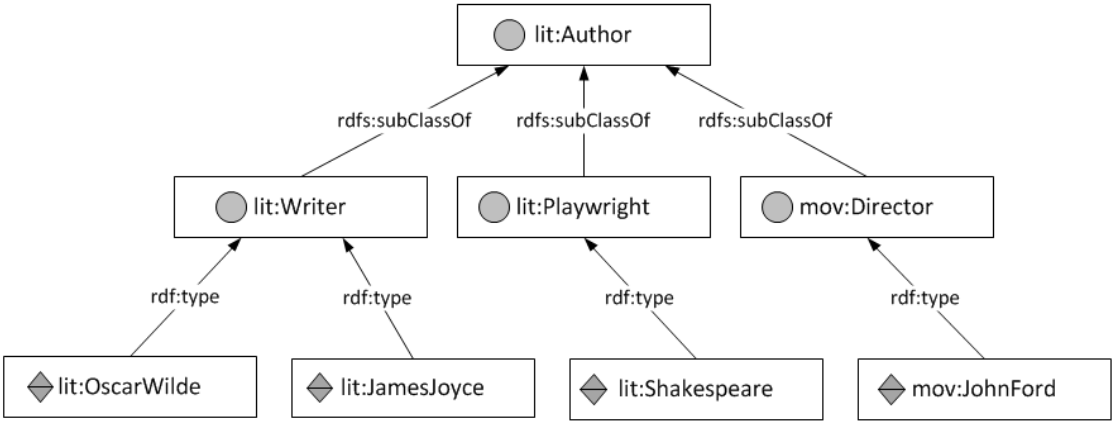
- **RDFS-Plus**. RDFS-Plus is a subset of OWL that includes enough expressivity to describe how certain properties can be used and how they relate to one another. RDFS-Plus is expressive enough but it lacks the complexity that makes OWL.

- **OWL (Web Ontology Language).** OWL brings the expressivity of logic to the Semantic Web. It allows modelers to express detailed constraints between classes, entities, and properties. OWL was adopted as a recommendation by the W3C in 2003.

"Smart data" example:

Turning to our previous example of Elizabethan Literature, we will see how to provide certain meaning to our relationship "Playwright" is a subcategory of "Author" by using RDFS modeling building blocks. We will use "rdfs:subclassOf" to express that "Playwright" is a subcategory of "Author" and "rdf:typeOf" to express that an entity corresponds to a certain category. To make thing more challenging, we will add movie data from the data source American Movies organization.

```
lit:Playwright rdfs:subclassOf lit:Author
lit:Writer rdfs:subclassOf lit:Author
mov:Director rdfs:subclassOf lit:Author
```



From the figure above, we can laid down that the **asserted triples** are

```
lit:OscarWilde rdf:type lit:Writer
lit:JamesJoyce rdf:type lit:Writer
lit:Shakespeare rdf:type lit:Playwright
mov:JohnFord rdf:type mov:Director
```

And we can also infer that the **inferred triples** are

```
lit:OscarWilde rdf:type lit:Author
lit:JamesJoyce rdf:type lit:Author
lit:Shakespeare rdf:type lit:Author
mov:JohnFord rdf:type lit:Author
```

Finally, we can define the following taxonomy: "Both Writer and Playwright are narrow terms from the broader term Author".

Table 3.8 shows different Semantic Web constructions classified by Semantic Web modeling language they belong to.

Table 3.8 Semantic Web modeling language constructions

	Construction	Description
RDF	rdf:type	The relationship between an instance and its type.
RDF	rdf:Property	The type of any property in RDF.
RDFS	rdfs:subClassOf	Relation between classes, that the members of one class are included in the members of the other.
RDFS	rdfs:subPropertyOf	Relation between properties, that the pairs related by one property are included in the other.
RDFS	rdfs:domain rdfs:range	Description of a property that determines class membership of individuals related by that property.
RDFS	rdfs:label	No inferential semantics, printable name
RDFS	rdfs:comment	No inferential semantics, information for readers of the model
OWL	owl:sameAs	All statements about one instance hold for the other.
OWL	owl:inverseOf	Exchange subject and object
OWL	owl:TransitiveProperty	Chains of relationships collapse into a single relationship.
OWL	owl:SymmetricProperty	A property that is its own inverse.
OWL	owl:FunctionalProperty	Only one value allowed (as object).
OWL	owl:InverseFunctionalProperty	Only one value allowed (as subject).
OWL	owl:ObjectProperty	Property can have resource as object.
OWL	owl:DatatypeProperty	Property can have data value as object.
OWL	owl:Restriction	The building block in OWL that describes classes by restricting the values allowed for certain properties.
OWL	owl:hasValue	A type of restriction that refers to a single value for a property.
OWL	owl:someValuesFrom	A type of restriction that refers to a set from which some value for a property must

		come.
OWL	owl:allValuesFrom	A type of restriction that refers to a set from which all values for a property come.
OWL	owl:onProperty	A link from a restriction to the property it restricts.
OWL	owl:unionOf owl:intersectionOf owl:complementOf	Basic set operations applied to classes. Each of these is used to create a new class, based on the specified set operation applied to one or more defined classes.
OWL	owl:oneOf	Specifies that a class consists just of the listed members
OWL	owl:differentFrom	Specifies that one individual is not owl:sameAs another. This is particularly useful when making counting arguments.
OWL	owl:cardinality owl:minCardinality owl:maxCardinality	Cardinality specifies information about the number of distinct values for some property. Combined with owl:oneOf, owl:differentFrom, owl:disjointWith,and so on, it can be the basis of inferences based on counting the number of values.

3.3. Semantic Web Application Architecture

3.3.1. The design of a Web Semantic Application

Once we have already seen the components of Semantic Web Technology Stack, it is time to discuss how to come up with a Semantic Web application and how these components fit together. The Semantic Web Architecture is shown and described in the Figure 3.13.

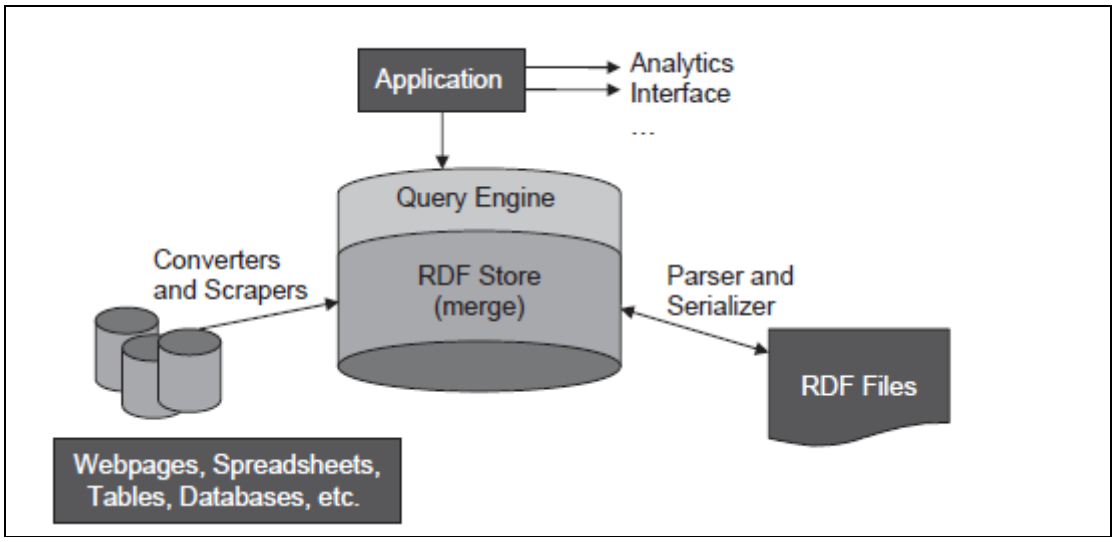


Figure 3.13 Semantic Web Architecture

- **RDF Parser/Serializer.** We have already seen a number of serializations of RDF in the Semantic Web Technology Stack, including the W3C standard serialization in XML. An RDF parser reads text in one (or more)

of these formats and interprets it as triples in the RDF data model. An RDF serializer does the reverse process; it takes a set of triples and creates a file that expresses that content in one of the serialization forms.

- **RDF Store.** An RDF store is a database that is tuned for storing and retrieving data in the form of triples. In addition to the familiar functions of any database, an RDF store has the additional ability to merge information from multiple data sources.

-

- **RDF Query Engine.** Closely related to the RDF store is the RDF Query engine. The query engine provides the capability to retrieve information from an RDF store according to structured queries.

- **Application.** An application has some work that it performs with the data it processes: analysis, user interaction, archiving, and so on. These capabilities are accomplished using some programming language that accesses the RDF store via queries (processed with the RDF query engine).

Most of these components have corresponding components in a relational database application as Table 3.9 below shows.

Table 3.9. Comparison between Relational databases and Semantic Web

	Relational Database	Semantic Web
Data Repository	RDBMS	RDF store
Query Engine	RDBMS Query Engine	RDF Query Engine
Data Definition Language	SQL	RDF
Query Language	SQL	SPARSQL
Parser/Searializer	No applicable	Turtle, NTriples, XML/RDF
Converter/Scrapper	No applicable	RDFa
Application	Custom code	Custom code

3.3.2. Adding inference to our Semantic Web Architecture

The inference we outlined in previous sections for describing the meaning of data is useful, but how does it fit with the architecture we have just defined? We will need to add a new component in our architecture, something that can respond to queries based not only on the triples that have been asserted but also on the triples that can be inferred based on patterns of inference.

The new component, which could be called Inference Query Engine, will be part of the RDF Query Engine and its power of inferencing should be determined by the set of inferences that it can support. For example an OWL Inference Query Engine supports a larger set of inferences than a RDFS one. As a result of this design, we actually have two different separated processes the Query Engine architectural component is responsible for. On one hand, we have a query process which can query all asserted triples as result of merging from several data sources. On other hand, we also have an inference process

which produces all the possible inferred triples based on a particular set of inference patterns.

You may be wondering why we give so much importance to this nuance. The process of inference leads to a very important question that can determine the performance of our system. When does inference actually happen? Where are inference triples stored?

- **Cache inference.** This approach stores all inference triples in the RDF store along with asserted triples. As soon as inference pattern is identified, any inferred triples are inserted into the RDF store. The disadvantage of this approach is that the triple store could be overloaded and affects performance.

- **Just in time inference.** This approach stores all inference triples in the RDF store only in response to queries only. The query responses are produced in such a way as to respect all the appropriate inferences, but no inferred triple is retained. The drawback of this approach is that inference work is duplicated but the persistent storage is eased.

3.4. A brief summary of Web Semantic concepts learnt

The aspects of the Semantic Web we outlined here give us a subtle idea of how unruly the Web can become. We could observe that Semantic Web deals with the same sort of problem we spelt out at the beginning of this document — that is, inconsistent, disconnected and out-of-synchronization data. How can multiple data sources be brought together? How can such a mess become something useful?

Semantic Web overcomes this challenge by several mediums called Semantic Web Stack: a representation of distributed web data, a process of merging data, some modeling tools to give sense to this mass of data, a novel query engine with support for inference, and even a design model for Semantic Web applications. Its mission is to make sensible, usable, and durable information resources.

In certain way, Semantic Web encourages us to carry out a reengineering process of reconverting our supposed organized data into raw data in such way to give them a new sense. This is the idea behind Semantic Web.

Table 3.10 shows a summarize of Semantic Web concepts

Table 3.10 Semantic Web concepts

Concept	Description
RDF	This distributes data on the Web.
Triple	The fundamental data structure of RDF. A triple is made up of a subject, predicate, and object.
Graph	A nodes-and-links structural view of RDF data.
Merging	The process of treating two graphs as if they were one.
URI	A generalization of the URL (Uniform Resource Locator), which is the global name on the Web.

qname	An abbreviated version of a URI, it is made up of a namespace identifier and a name, separated by a colon
rdf:Type	The relationship between an instance and its type.
rdf:Property	The type of any property in RDF.
rdf:SubclassOf	The inheritance relationship in RDFS.
N-triples, N3, RDF/XML	The serialization syntaxes for RDF.
Ontology	A way to make date sensible distributed data
Taxonomy	A way to provide a common understanding on a domain
Asserted triples	The triples in a graph that were provided by some data source.
Inferred triples	Triples that were added to a model based on systematic inference patterns.
Inference patterns	Systematic patterns defining which of the triples should be inferred.
Inference engine	A program that performs inferences according to some inference rules. It is often integrated with a query engine.
RDF parser/serializer	A system component for reading and writing RDF in one of several file formats.
RDF store	A database that works in RDF. One of its main operations is to merge RDF stores.
RDF query engine	This provides access to an RDF store, much as an SQL engine provides access to a relational store.
RDF inference query engine	This provides access to an RDF store, much as an SQL engine provides access to a relational store.
SPARQL	The W3C standard query language for RDF.
SPARQLU	Extension for SPARQL to perform CRUD operations
SPARQL endpoint	Any application that can answer a SPARQL query, especially one where the native encoding of information is not in RDF.
Application interface	The part of the application that uses the content of an RDF store in an interaction with some user.
Scraper	A tool that extracts structured information from webpages.
Converter	A tool that converts data from some form (e.g., tables) into RDF
RDFa	Proposed standards for encoding and retrieving RDF metadata from HTML pages.

PART III

“As a general rule, the most successful man in life is the man who has the best information.” —Benjamin Disraeli

4. The journey from Semantic Web to Enterprise Semantic Web

Semantic Web can provide an interesting approach for organizations to deal with inconsistent, disconnected and out-of-synchronization data. In fact, we will not be pioneers in this field because there is already a hard proof of its feasibility.

Success Case in organizations: Data.gov

Data.gov is a successful effort made by the US government to publish public information from different organizations. There are thousands of data sets in Data.gov, of which hundreds are made available in RDF. Data.gov is a great example of how to put order in chaos; the published data sets come from a wide variety of data sources with different formats and methodologies. Data.gov showed how Semantic Web can be useful to organizations at once. So, if they could, we can!

From now on, our mission will be to rename Web Semantic to Enterprise Semantic Web. We will look into how SharePoint, an enterprise solution, can be blend into Web Semantic as a knowledge management platform within organizations.

Calm your horses! If we want to go deeper into Enterprise Semantic Web through SharePoint, first of all, we need to have a brief look at SharePoint architecture so as to understand our subsequent steps. Secondly, we will dive to the depths of SharePoint data model to find out how SharePoint data model can match with the Semantic Web representation of data. Thirdly, we will keep on investigating into how SharePoint brings all data together and gives sense to those data. Finally, we will have a brief look at querying data. Only then, we will surface. So now let's get down to business.

4.1. SharePoint Architecture

At its core, SharePoint is a data provisioning engine —that is, its fundamental design is based on the idea of using Web-based templates to create sites, lists, and libraries to store and organize data.

SharePoint is particularly helpful to companies and organizations faced with the task of creating and administering a large number of websites along with its data. Someone in the IT department or even an ordinary business user can provision a site in SharePoint in less than a minute.

4.1.1. Server Farms

Every deployment of SharePoint is based on the concept of a farm. A server farm is a set of one or more server computers working together. A SharePoint

farm in a typical production deployment runs several Web servers, Application servers and Database servers coming together to provide Web Application and Services Applications to users.

The architecture of SharePoint was specifically designed to operate in a Web farm environment. Figure 4.1 shows a diagram of a typical Web farm. Our remainder description of SharePoint architecture will be based on this diagram.

Scenario: Introducing Contoso

Many of the examples in this document are based on Contoso also known as Contoso Ltd., a fictional company used by Microsoft as an example company and domain. Contoso has a long and proud history of manufacturing and delivering its products in bulk for all its customers. Recently, Constoso has recently set up an intranet using SharePoint to provide a means of collaboration between its remote and internal employees. Constoso has also rolled out an extranet using SharePoint to interact with partners around the world. Finally, Contoso decided to use SharePoint to erect its Internet-facing site to advertise and promote its products for customers.

Typical SharedPoint Design: Contoso Portal

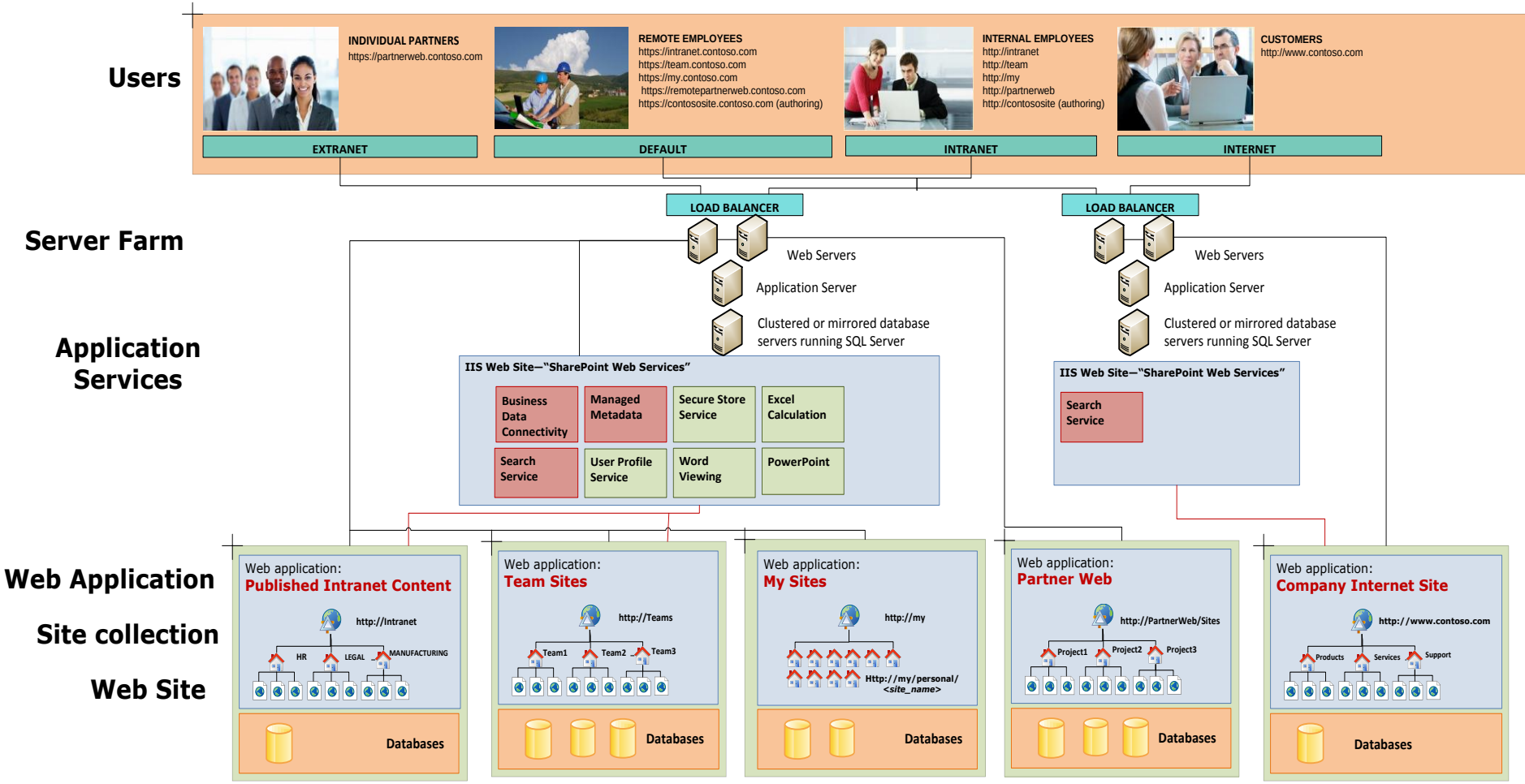


Figure 4.1. SharePoint Design Sample

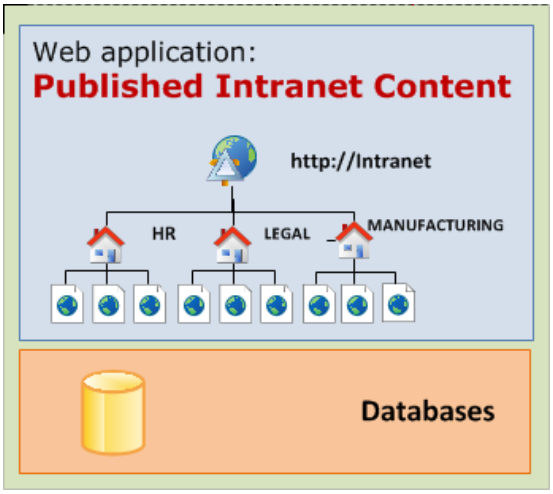
4.1.2. Web Applications

SharePoint is built on IIS Web server and relies on IIS websites to handle incoming HTTP requests. We can create additional IIS websites to provide additional HTTP entry points using different port numbers, different IP addresses, or different host headers. At a physical level, a SharePoint Web application is a collection of one or more IIS websites configured to map incoming HTTP requests to a set of SharePoint sites. The Web application, in turn, maps each SharePoint site to one or more specific content databases. SharePoint uses relational databases to store site content such as lists, list items, documents, and customization information.

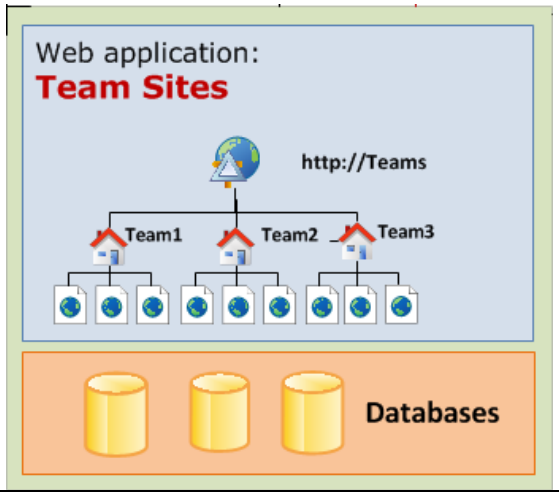
Scenario: Web Applications in Contoso

In our scenario, we will use host headers to create HTTP entry points for domain names such as *http://intranet.contoso.com*. SharePoint creates an abstraction on top of IIS that is known as a Web application. Contoso hosts the following Web applications within the server farm:

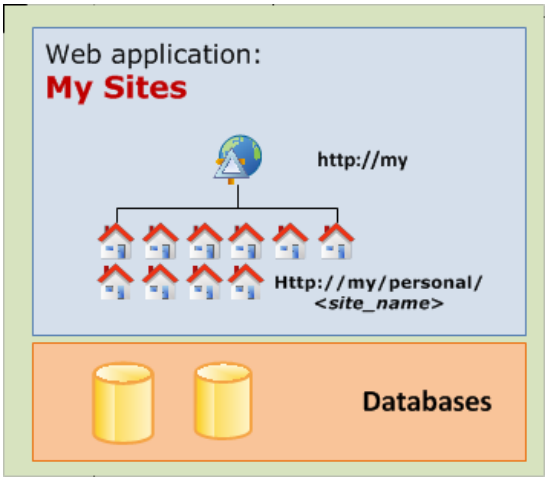
Published Intranet Content: Web sites that allow different internal departments to host content like documents, reports, images, videos or even Web pages…



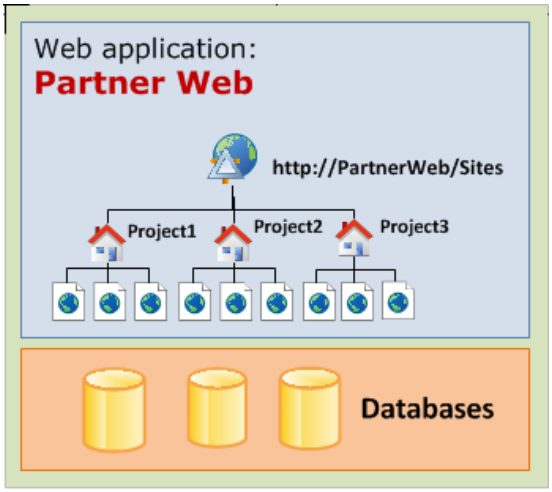
Team Sites: Web sites that allows organization members to work together on Microsoft Office documents which are stored an accessed by internal and remote employees.



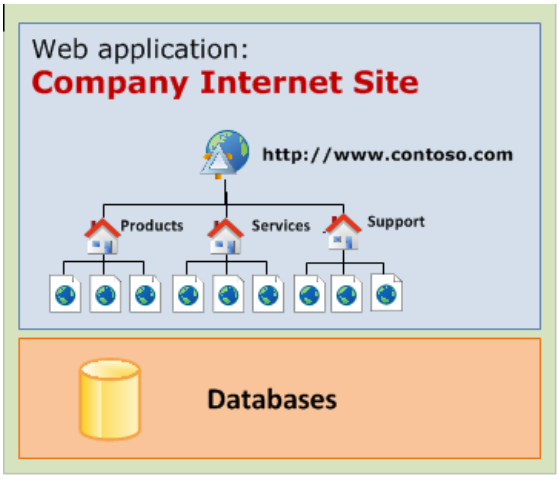
My Sites: My Sites are personal sites that not only display information about each user in the organization, but also are used as a personal landing page and storage site for individuals. My Sites can be used to enter information about yourself, such as demographics, current projects, areas of expertise, and so on.



Partner Web: Those project or services for which are necessary to work closely with external collaborators are hosted in isolated Web Applications.



Company Internet Site: This is the typical company Web Site whereby the organization gets into communication with customers to offer their products and services.

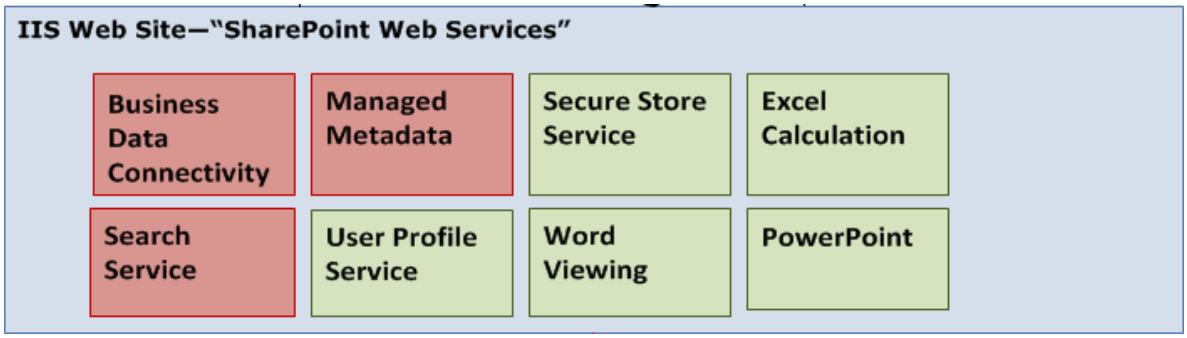


4.1.3. Service Applications

Service applications provide SharePoint functionality and share resources across sites running in different Web applications and different farms. SharePoint ships with many built-in service applications like Access Services Enterprise, Application Registry Services Standard… Each of them provides interesting functionality but we will not give them much importance since they are beyond scope of this document.

Scenario: Service Applications in Contoso

In our scenario, Contoso hosts several Service applications within the server farm but we will mainly have a focus on Business Connectivity Service, Managed Metadata Service and Search Service. These four key service applications are worthy of our interest because all of them are closely related to Semantic Web principles.



- **Business Connectivity Service** — to allow an organization to connect SharePoint-based solutions to sources of external data.
- **Managed Metadata Service** — to allow an organization, team or department to manage its own taxonomy, hierarchies, keywords, and so on.
- **Search Service** — to query, index and crawl content and users.

4.1.4. Web site

Now that we understand the high-level architecture of a server farm along with Web applications and Service applications, we need to have a look at how SharePoint creates and manages sites within the scope of a Web application.

What exactly is a SharePoint Web site? A Web site is an endpoint that is accessible from across a network such the Internet, an intranet, or an extranet. A site is also a storage container that allows users to store and manage content such as list, document libraries, list items and documents.

Figure 4.2 shows some Web sites hosted in the Web application "Published Intranet Content" in the Contoso Server farm.

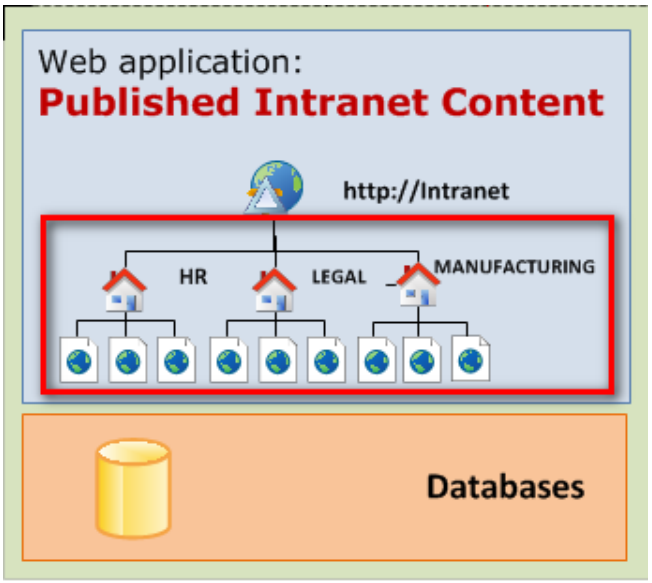


Figure 4.2. Web Sites Contoso Sample

4.1.5. Site Collection

Every Web site must be provisioned within the scope of an existing Web application. However, a Web site cannot exist as an independent entity within a Web application. Instead, every Web site must also be created inside the scope of a site collection. A site collection is a container of sites. Every site collection has a top-level site from child sites derive.

You might be asking yourself why the SharePoint architecture requires this special container to hold its sites. Site collections represent a scope for administrative and security settings. All sites within a site collection maintain the same administrative and security settings except for some custom setting at Web site level. Think about the requirements of site management in a large corporation that's provisioning thousands of sites per year. Figure 4.3 shows a site collection hosted in the Web application "Published Intranet Content" in the Contoso Server farm.

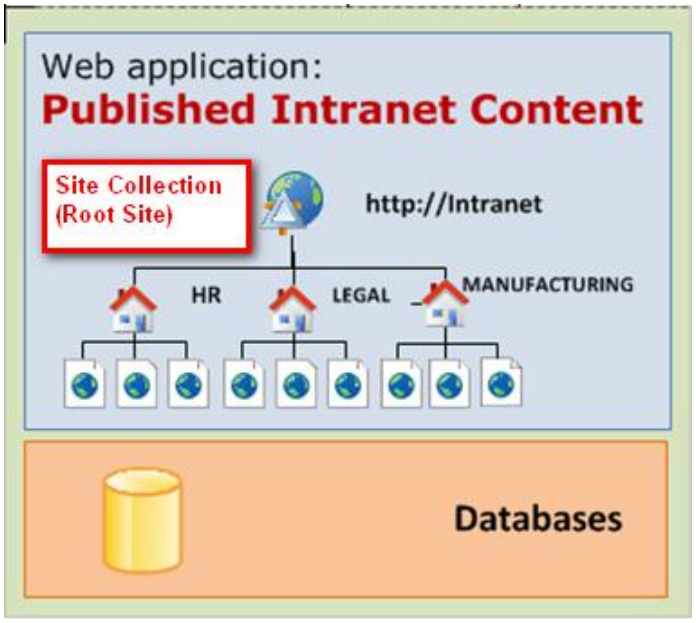


Figure 4.3. Site Collection Contoso Sample

4.1.6. Databases

SharePoint uses content relational databases to store site content such as lists, list items, documents, and customization information. The Web application also maps each SharePoint site to one or more specific content databases and each site collection is stored in only single content databases associated with a Web application. A typical SharePoint server farm could easily reach up to 50 or more content databases without counting databases associated with server farm configuration and service applications.

Scenario: Don't Touch the SharePoint Databases

We could yield to temptation of directly handling SharePoint databases to resolve our Semantic Web issue but this is not possible. When developing for SharePoint, we are not permitted to directly access the configuration database or any of the content databases. For example, we must resist any temptation to write custom code that reads or writes data from the tables inside these databases. Instead, we should write code against the SharePoint programming APIs to reach the same goal, and leave it to SharePoint to access the configuration database and content database behind the scenes. This restricts fully the developers to develop custom applications.

4.2. SharePoint Data Model

.

SharePoint distinguishes two different scenarios for data model design. In one scenario, there is no existing system to worry about and we can start our internal data model design from scratch. This situation is known as green field

development. In the other situation, we have to design the solution in an existing environment, which could include integration with external systems, legacy systems and existing databases. This other situation is known (please, no jokes!) as brown field development.

4.2.1. Green field development

Every custom SharePoint application is a data-driven application in one way or another and it provides a data model very close to relational database model. SharePoint is also built on Oriented-Object programming framework and sometimes it can be seen as an Object-Relational mapping framework. Table 5.1 shows SharePoint data model building blocks that are conceptually similar to those found in a relational database and Semantic Web.

SharePoint, Relational database and Semantic Web		
SharePoint	**Relational Database**	**Semantic Web**
List	Table	Set of Triples
View	Table View	rdfs:domain, rdfs:range
Column	Table Column	rdf:property
List Item	Table Row	Triple
Content Type	Table Schema	rdf:type
Content Type Hierarchy	No applicable	rdfs:subclassOf
No applicable	No applicable	rdfs:subpropertyOf
Lookup column	Foreign key	Graph

4.2.1.1. SharePoint Column

A column represents a reusable metadata that we can be assigned to multiple content types across multiple Web sites within a Site collection. SharePoint columns are conceptually similar to table columns in that they define data type and other features columns will store.

SharePoint columns and RDF properties

A RDF property indicates when other identifier is to be used as a predicate in a triple in such way that it can be assigned to different types to build sentences like “The order number ZX-3P (subject) costs (predicate) 1.150 € (object)”. Similar to RDF, SharePoint columns can be assigned to different content types and build the same sentence: “The order number ZX-3P (content type) costs (column) 1.150 € (value)”.

SharePoint columns and RDFS range and domain

RDFS domain asserts that a RDF property relates values from a certain domain whereas RDFS range establishes that a range of values relates a RDF property. A simple example of RDFS domain in SharePoint could be that a SharePoint column “OrderNo” relates basic data type like integer or float. This example is pretty simple and it can be easily applicable to SharePoint columns.

Look out for the combination of RDFS domain and RDFS class inheritance

A combination between RDFS domain and RDFS class inheritance can be very confusing to understand. For example, by using RDF domain we can assert "If an order inherits from a shippable item and that order has a property called identification whose domain is Order, then we can infer that any item with property identification must be an Order, what's more, it must be also a shippable item". This sort of assertions is not possible in SharePoint columns. The reason is that SharePoint is built on an Oriented-Object framework. In the Semantic Web, because of the slogan "AAA", a RDF property can be used anywhere, and it must be independent of any class. That is, it is never accurate in the Semantic Web to say that a property is "defined for a class". A property is defined independently of any class, and the RDFS relations specify which inferences can be made. To sum up, RDFS domain and RDFS range have their counterparts in SharePoint but to a certain extent as long as they are not used in combination with inheritance.

4.2.1.2. SharePoint Content type

A content type is a reusable collection of metadata (columns) for list items in a list. Content types enable you to manage the settings for a category of information in a centralized and reusable way within a Site collection. In relational database, the only similar concept to content type is a schema created by Definition Data Language (DDL) for a database table.

SharePoint Content types and RDF type

A RDF type is the property that provides an elementary typing system in RDF in the same way as is content type in SharePoint. In RDF, we can say "The order number ZX-3P (subject) is of the type (predicate) Order (object)". In SharePoint, we can equally say "The order number ZX-3P (instance of content type) is of the type (content type associated with list item object) Order (content type)".

SharePoint Content type hub

We may have let a little nuance escape in the previous definition of content type before. A content type is defined within a Site collection, but what if we needed the content type to be used in multiple Site collections. If we need to use content types across multiple Site collections, we would have to set up a Content Type hub in Managed Metadata Service Application about which we will speak soon. This Content Type hub will be responsible for managing content types that can be published across all Web application and its Site collections.

4.2.1.3. SharePoint Content Type hierarchy

SharePoint includes many built-in content types that can be organized in a hierarchy based on inheritance. At the top of the hierarchy is the System content type which is composed of important built-in columns such as ID. Next below System is the Item content type, which is derived from System. Item is made up of only one built-in important column called title that is normally used to store the name associate with any a content type. All built-in content types in SharePoint ultimately inheritance from Item. In SharePoint, there are many content types defined such as Task, Contact, Event and so on and on forth. Before creating a new content type from scratch, you should wonder whether it is really worth creating a new content type or extend an existing content type adding new columns.

SharePoint Content types inheritance and RDFS subclasses

RDFS subclass suits SharePoint content type inheritance very well. In fact, although SharePoint Content type is built on Oriented-object framework, it matches better with the concept of inheritance in RDFS.

In Oriented-object programing (OOP, for the sake of brevity), an instance of a class responds to the same methods in the same way as instances of its superclass. In SharePoint terms, this is because that instance is also a member of the superclass, and thus must behave like any such member. For example, the reason why an instance of class “Document” responds to methods defined in “System” is because the instance actually is also a member of class “Document.” This consistency is misleading when, in the OOP system, the subclass defines an override for a method defined in the superclass. In SharePoint terms, the instances of “Document” are still instance of “System” and should respond accordingly. But in most OOP semantics, this is not the case; the definitions at “Document” take precedence over those at “System” and thus “Document” need not actually behave like “System” at all. In the logic of the SharePoint, this is not allowed, there is no polymorphism.

SharePoint Content types inheritance and RDFS subproperties

Unfortunately SharePoint lacks the inference pattern RDFS subproperty. It is a pity indeed since SharePoint columns are independent of any content type. When we assign columns to content types, what we are actually doing is to make a copy from the original column and to assign it to the content type – that is, columns behaves like class-object relationship in OOP. However, it is not simply right in SharePoint to say that a lonely column has meaning by itself because it is useless unless they are associated with content types.

Warning: Lists with columns or lists with content types

Although it is possible to define a data model using only lists and columns, the recommended approach is to use content types to define your entities in SharePoint. The main reason behind this recommendation is those list items that are stored in a list with no content type associated with are not actually typified. For example, if there are some contact list deployed in SharePoint and those lists have a contact content type associated, you will only have to modify the content type if a change was required. However, if those contact lists have no content type associated, you will have to go across all contact lists to make the new changes. A pain in the neck!

4.2.1.4. SharePoint List and List item

Lists are the storage mechanism in SharePoint and conceptually (watch what we say conceptually) similar to a relational database table in that, they are made of columns and rows, that we can define primary keys, and that we can create relationships between lists. Lists stores a set of rows called list items. Each list item is typified by a content type and it can see as an instance of a content type. Conversely, a content type defines the structure of multiple list items.

SharePoint Lists Item and RDF triples

A RDF triple corresponds to a piece of list item. For example, if in RDF we say "The order number ZX-3P (subject) costs (predicate) 1.150 € (object)", this sentence corresponds to the SharePoint list item with "Order number" column corresponding to the value "ZX-3P" whose "Cost" column holds the value "1.150 €".

An entire set of RDF triples describing a same resource corresponds to an entire list item. If we have several triples related to the same resource such as "The order number ZX-3P costs 1.150 €" and "The order number ZX-3P has an amount of 23 items", these sentences correspond to the SharePoint list item whose "Order number", "Cost" and "Amount" columns hold the values "ZX-3P", "1.150 €" and "23" respectively.

How the problem of unique identity is resolved in SharePoint

In order to provide an identity to SharePoint objects, SharePoint provides every list and list item with an identity (ID). A list is uniquely identified by Web Application, Web site and its list ID. If we want to identify a list item, we only have to add its ID to the Web Application, Web site and list.

For example, a product list in the Contoso Manufacturing department can be identified by the URN http://intranet.contoso.com/manufacturing/ProductList in which http://intranet.contoso.com corresponds to the Web application that host the list and the relative URN /manufacturing refers to the Web that host the list. In case we want to identify a certain product (a list item) stored in the product list, we should only add its ID to the URN.

As you can see, the URN Web Application/Site Collection/Web/List(/ID) is pretty similar to the concept of URN in Semantic Web. Unfortunately, no qnames are provided by SharePoint.

SharePoint Lists and several RDF triples

Many sets of RDF related to the same type of resource that are stored in a RDF Store correspond to the concept of SharePoint list. Keep in mind that a SharePoint list is stored in a data repository as a set of RDF triples.

A SharePoint List corresponds to a relational data table in content database?

Thinking that there is a straight correlation between a SharePoint list and relational database table in a content database is a typical mistake in SharePoint. When we defined the relationship between a SharePoint list and relational database table, we stressed that such relationship was actually conceptual rather than physical. How is a SharePoint list actually stored in relational database? We are afraid that it is a Microsoft mystery like many others. SharePoint does not provide this sort of support. If you ask, you will receive a gentle advice in exchange: Don't touch databases!! What it is sure is that a neither SharePoint list is necessarily to be related to a single database table nor SharePoint list content is handled by database. For example, A SharePoint list can be indexed to enhance its performance and such index is not stored in the database but in the system server files where Web application is deployed. This pulls down any theory of physical relationship with database tables.

4.2.1.5. SharePoint List View

A list view is conceptually similar to its relational database view counterpart. We can use views on a list to sort, group, and filter list items. There is no a Web Semantic building block similar to the concept of view in SharePoint, SPARQL allows us to define views though. The concept of SharePoint view is tremendously important when we must deal with large lists later, thus we are introducing this concept.

4.2.1.6. Putting columns, content types and lists together

It is time we put together all SharePoint data model building block to show how SharePoint data model can be used in RDF. We will be turning to our company Contoso to produce a RDF graph that reflects the content of some SharePoint lists in such a way that the information is preserved and amenable for RDF operations like merging and querying. Suppose that given the orders list that is stored in the Web site http://intranet.consoso.com/manufacturing as it is shown below

Orders SharePoint List			
ID (built in)	**OrderNo**	**Amount**	**Cost**
1	ZX-3	23	1.150 €
2	ZX-3P	4	120 €
3	ZX-3S	34	6.550 €
4	B-1430	23	875 €
5	B-1430-X	14	300 €

Each list item in the list describes a single entity, all of the same type. That type is given by the name of the list itself, Order. We know certain information about each of these items, based on the columns in the list itself (remember that there is always a content type behind a list), such as Order NO, Amount, and so on. We want to represent this data in RDF. Since list item row represents a distinct entity, each list item will have a distinct URI. Fortunately, as we see the need for unique identifiers is just as present in SharePoint as it is in the Semantic Web, so there is a (locally) unique identifier available—namely, the primary list key, in this case the built-in column called ID.

For the Semantic Web, we need a globally unique identifier. The simplest way to form such an identifier is by having a single URI for the list itself. Remember that a list item can be identified by the triple Web Application/Web Site/List plus its ID (for example, http://intranet.contoso.com/manufacturing/orders#1. We use that URI as the namespace for all the identifiers in the list for the sake of brevity. Since this is a Web site for manufacturing, let's call that namespace mfg:. Then we can create an identifier for each order by concatenating the table name "Order"

with the unique key and expressing this identifier in the mfg: namespace, resulting in identifiers mfg:Order1, mfg:Order2, and so on.

Each list item in the list says several things about that list item—namely, its Order number, amount, and so on. To represent this in RDF, each of these will be a property that will describe the Orders. But just as is the case for the unique identifiers for the rows, we need to have global unique identifiers for these properties. We can use the same namespace as we did for the individuals, but since two list could have the same column name (but they aren't the same properties!), we need to combine the list name and the column name. This results in properties like mfg:Order_OrderNo, mfg:Order_Amount, and so on.

With these conventions in place, we can now express all the information in the list as triples. There will be one triple per list item in the list—that is, for n rows and c columns, there will be n * c triples. The data shown in Table 5.2 has 4 columns and 5 rows, so there are 20 triples, as shown in Table 5.3 below.

Table 5.3 Triples representing data stored in Orders SharePoint list

Subject	Predicate	Object
mfg:Order1	mfg:Order_ID	1
mfg:Order1	mfg:Order_OrderNo	ZX-3
mfg:Order1	mfg:Order_Amount	23
mfg:Order1	mfg:Order_Cost	1.150 €
mfg:Order2	mfg:Order_ID	2
mfg:Order2	mfg:Order_OrderNo	ZX-3P
mfg:Order2	mfg:Order_Amount	4
mfg:Order2	mfg:Order_Cost	120 €
mfg:Order3	mfg:Order_ID	3
mfg:Order3	mfg:Order_OrderNo	ZX-3S
mfg:Order3	mfg:Order_Amount	34
mfg:Order3	mfg:Order_Cost	6
mfg:Order4	mfg:Order_ID	4
mfg:Order4	mfg:Order_OrderNo	B-1430
mfg:Order4	mfg:Order_Amount	23
mfg:Order4	mfg:Order_Cost	875 €
mfg:Order5	mfg:Order_ID	5
mfg:Order5	mfg:Order_OrderNo	B-1430-X
mfg:Order5	mfg:Order_Amount	14
mfg:Order5	mfg:Order_Cost	300 €

The triples in the table are a bit different from the triples we have seen so far. Although the subject and predicate of these triples are RDF resources (complete with qname namespaces), the objects are not resources but literal data—that is, strings, integers, and so forth. This should come as no surprise, since, after all, RDF is a data representation system.

The usual interpretation of a list is that each list item in the list corresponds to one individual and that the type of these individuals corresponds to the

name of a content type. In Table 5.3, each list item actually corresponds to an Order content type. We can represent this in RDF by adding one triple per list item that specifies the type of the individual described by each list item, as shown in Table 5.4.

Table 5.4 Triples representing Type information

Subject	Predicate	Object
mfg:Order1	rdf:type	mfg:Order
mfg:Order2	rdf:type	mfg:Order
mfg:Order3	rdf:type	mfg:Order
mfg:Order4	rdf:type	mfg:Order
mfg:Order5	rdf:type	mfg:Order

Wait a moment! For what we have Content type inheritance? Should the Order content type not inherit from Item content type as we defined before? What about the properties? Table 5.5 shows the entire representation of data.

Be aware that we have introduced a new qname corresponding to the SharePoint Item content type called sp: in honor of SharePoint.

Table 5.5 Triples representing entire information

Subject	Predicate	Object
mfg:Order_ID	rdf:type	rdf:property
mfg:Order_OrderNo	rdf:type	rdf:property
mfg:Order_Amount	rdf:type	rdf:property
mfg:Order_Cost	rdf:type	rdf:property
mfg:Order_ID	rdfs:domain	xsd:string
mfg:Order_OrderNo	rdfs:domain	xsd:string
mfg:Order_Amount	rdfs:domain	xsd:integer
mfg:Order_Cost	rdfs:domain	xsd:decimal
mfg:Order	rdfs:subClassOf	sp:Item
mfg:Order_ID	rdfs:subPorperttyOf	sp:ID
mfg:Order1	rdf:type	mfg:Order
mfg:Order2	rdf:type	mfg:Order
mfg:Order3	rdf:type	mfg:Order
mfg:Order4	rdf:type	mfg:Order
mfg:Order5	rdf:type	mfg:Order

From the table above, we can infer the following inferred triples

```
mfg:Order1 rdf:type sp:item
mfg:Order2 rdf:type sp:item
mfg:Order3 rdf:type sp:item
mfg:Order4 rdf:type sp:item
mfg:Order5 rdf:type sp:item
```

4.2.1.7. Lookup column and relationships between lists

SharePoint allows you to create relationships between lists through a special type of column in SharePoint call lookup column. The great benefit of this functionality is that SharePoint allows us for list to be joined by SharePoint queries languages. This ability to query across lists brings SharePoint data models very close to relational database model.

In order to explain how to create relationships in SharePoint lists, we will turn to our company Contoso again – namely our Orders and Order Lines lists that are stored in the Web site http://intranet.consoso.com/manufacturing. We shall start with describing relational database tables and after we will go on to introduce SharePoint and Semantic Web relationships.

Unrelated relational database tables

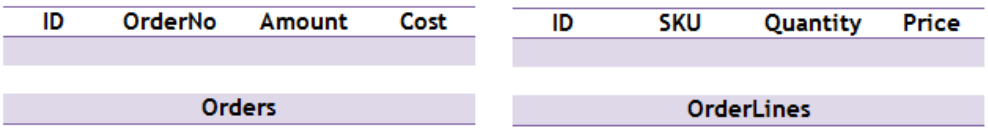


Figure 5.3 Unrelated database tables

Database tables linked by foreign key constraint (primary key)

To relate the tables, you could add an OrderID column to the OrderLines table, and use this column to define a foreign key relationship between the tables.

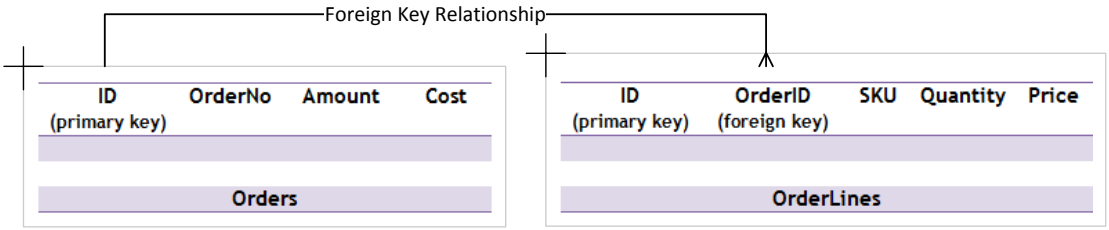


Figure 5.4 Database tables linked by foreign key constraint (primary key)

Database tables linked by foreign key constraint

Alternatively, you could add an OrderNo column to the OrderLines table, and use this column to define the foreign key relationship (providing that the OrderNo column in the Orders table is subject to a unique values constraint).

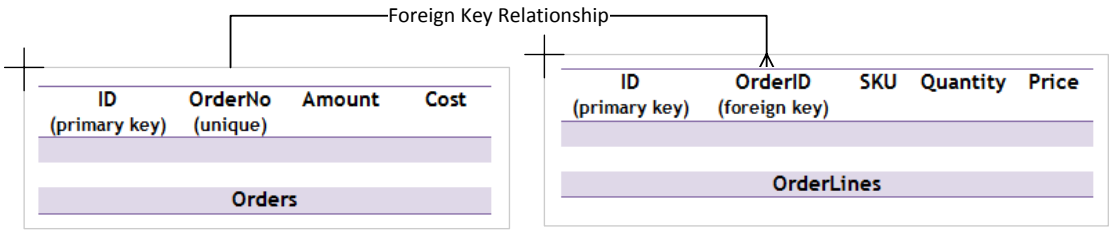


Figure 5.5 Database tables linked by foreign key constraint

Lookup column relationship between SharePoint lists

Lookup column relationships in SharePoint are conceptually similar to foreign key constraints in relational databases, but there is a key difference. Imagine we want to implement the previous example in a SharePoint data model. When we enter a new order line to the Order Lines list, we select the associated Order using the lookup column. SharePoint does not permit us to handle the foreign key constraint because the built-in ID column (inherited from content type System) is a sacred and unchangeable primary key that is internally handled. We should instead choose the column in the target list that we want to display in the source list, by setting up the list (ShowField attribute). SharePoint establishes the foreign key constraints for us.

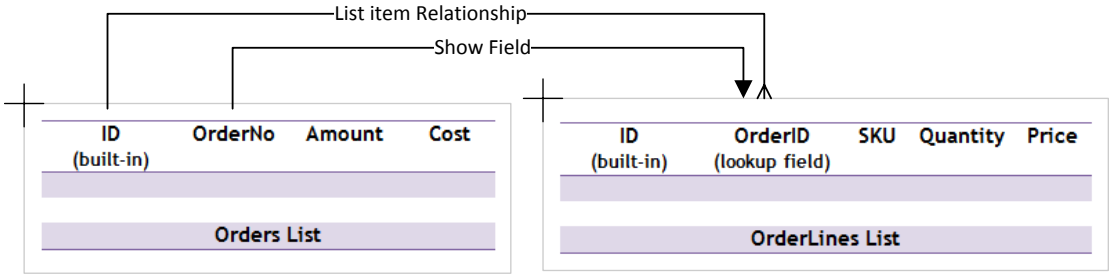


Figure 5.6 Lookup column relationship between SharePoint lists

many-to-many relationship between SharePoint lists

In case we want to model a many-to-many relationship using lists, you must create an intermediate list to normalize the relationship in the same way that you should do in relational database design. For example, suppose we want to model the relationship between parts and machines. A part can be found in many machines, and a machine can contain many parts. To normalize the relationship, we would create an intermediate list named PartMachine. However, from a user experience point of view, this is less than ideal, so at this point, we would probably add custom code to maintain associations between parts and machines rather than maintaining the intermediate list. But keep in mind that we can only handle show view fields not primary keys.

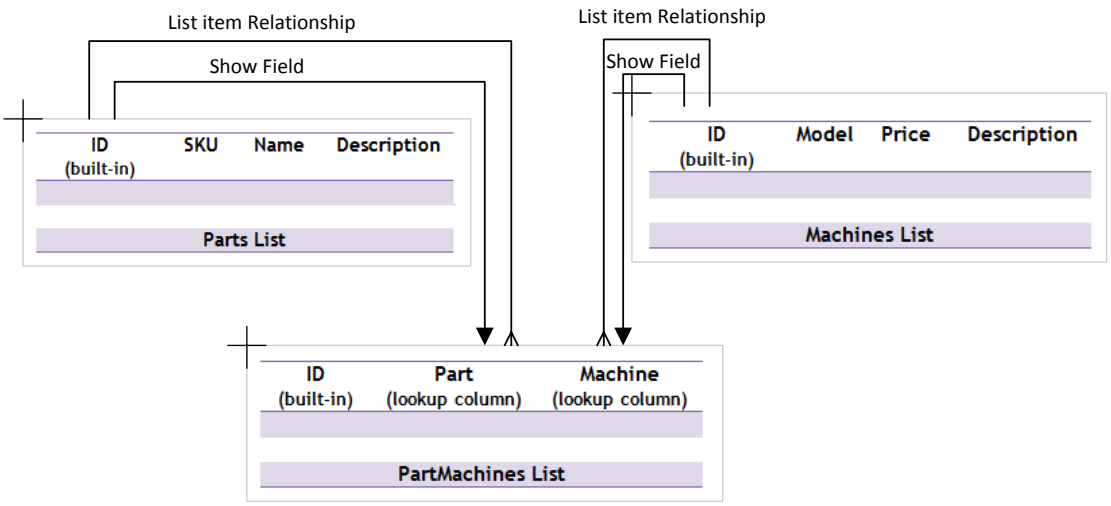


Figure 5.7 many-to-many relationship between SharePoint lists

Data relationships in Semantic Web

Leaving aside merging distributed data, the mission of Semantic Web is to combine data by defining concepts and relationships. Thinking that RDF would enable us to establish relationships very easily with a minimum of restrictions are fairly logical. Anything can be related to anything else with the aim to support the AAA-slogan (Anyone can say anything anytime).

Turning to Contoso, producing a RDFS graph reflecting relationships of some SharePoint lists must be a walk in the park. Imagine that in Contoso, some people are directly employed by the firm, whereas others are contractors. Among these contractors, some of them are directly contracted to the company on a freelance basis, and others contract through an intermediate firm. All of these people could be said to work for the Contoso. Nonetheless, RRHH department in Contoso stores several employee lists different to each other. Some of them were created by the RRHH department whereas the rest of them were created by the procurement department. Contoso management would like to establish some relationships between those employees' lists.

How can we model this situation in RDFS? First, we need to consider the inferences we wish to be able to draw and under what circumstances. There are a number of relationships that can hold between a person and the firm; we can call them contractsTo, freeLancesTo, indirectlyContractsTo, isEmployedBy, and works For. If we assert any of these statements about some person, then we would like to infer that that person worksFor Contoso. Furthermore, there are intermediate conclusions we can draw—for instance, both a freelancer and an indirect contractor contract to the firm and indeed work for the firm. All these relationships can be expressed in RDFS using the rdfs:subPropertyOf relation:

```
:freeLancesTo rdfs:subPropertyOf contractsTo.
:indirectlyContractsTo rdfs:subPropertyOf contractsTo.
:isEmployedBy rdfs:subPropertyOf worksFor.
:contractsTo rdfs:subPropertyOf worksFor.
```

To see what inferences can be drawn, we will need some instance data:

```
Goldman isEmployedBy TheFirm.
Spence freeLancesTo TheFirm.
Long indirectlyContractsTo TheFirm.
```

The rule that defines the meaning of rdfs:subPropertyOf implies a new triple, replacing any subproperty with its superproperty. So, since

```
isEmployedBy rdfs:subPropertyOf worksFor.
```

we can infer that

```
Goldman worksFor TheFirm.
```

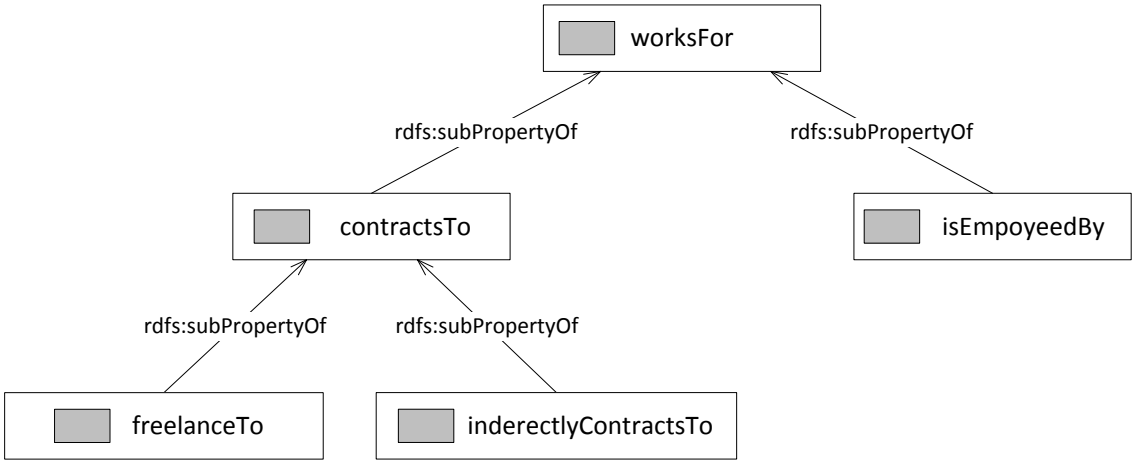
And because of the assertions about freelancing and indirect contracts, we can infer that

```
Spence contractsTo TheFirm.
Long contractsTo TheFirm.
```

And finally, since, like asserted triples, inferred triples can be used to make further inferences, we can further infer that

```
Spence worksFor TheFirm.
Long worksFor TheFirm.
```

In general, rdfs:subPropertyOf allows us to describe a hierarchy of related properties. Just as in class hierarchies, specific properties are at the bottom of the tree, and more general properties are higher up in the tree. Whenever any property in the tree holds between two entities, so does every property above it.



As we have just seen, establishing relationships between entities properties in Semantic Web is incredibly easy but the example above is not possible in SharePoint. Having lookup columns mess around, we can relate only entities with lookup columns. What about relating properties? Forget it because it is not possible since it lacks this sort of inference pattern. Our hopes of relating anything in SharePoint have just been shattered but we don't despair yet because SharePoint remains quite useful to our objectives.

WW3 gives us a hand

WW3 published some interesting documents. As they are good guys and understand relational databases keeps on underpinning most business, government and scientific enterprises, they propose some recommendations to make our relational data RDF data so as to, once translated, we can combine with data from other sources on the Web. The documents are "A Direct Mapping of Relational Data to RDF" and "R2RML: RDB to RDF Mapping Language".

4.2.2. Brown field development

If you have a memory like a sieve, do not worry because we will again put you in the picture of what brown field development means again. Brown field development is a scenario in which there are some existing systems such as external systems, legacy systems and existing databases. The mission is to build our data model design based on those systems and integrate those systems with your SharePoint data model. As we can see, this scenario is pretty similar to the principles of Semantic Web in which all data from multiple data sources are put together into a single data repository.

4.2.2.1. Business Data Connectivity Service

In SharePoint, the functionality that enables us to work with external data is provided by Business Connectivity Services (BCS, for the sake of brevity) that

is a Service Application deployed in our Server farm to provide service to all Web applications and, in turn, their Web sites.

When you develop a SharePoint application around external systems, we must design external data model similar to typical relational data models that are stored and managed by the BCS service application. This is known as a Business Data Connectivity model (BDC model, for the sake of brevity). In a BDC model, data entities are represented by external content types. An external content type models an external data entity, such as a database table or view or a web service connecting to an external system like an ERP, and defines a set of stereotyped operations on that data entity. In addition to external content types, a BDC model typically includes an external data source definition, connection and security details for the external data source, and associations that describe the relationship between individual data entities.

4.2.2.2. External Content Types and External lists

An external content type (ECT) is conceptually similar to a regular SharePoint Content type. Just like a regular content type, an ECT is a collection of metadata that models a data entity. External data sources describe data schemas in different ways. For example, Web services describe their data entities and operations using the Web Service Description Language (WSDL), while relational databases describe their data entities and operations using a database schema. When you create a BDC model in SharePoint, these data definitions are translated into ECTs.

An external list is a BCS component that provides a SharePoint list wrapper for data entities modeled by an ECT. The external list enables users to view, sort and filter, create, update, and delete external data entities in the same way that they would work with data in a regular SharePoint list.

4.2.2.3. Brown and green field development together

The key question of Business Connectivity Services (BCS) lies on Bussiness Data Connectivity Model (BDC). Developers do not actually have access to physical data from SharePoint, to put it clearer, developers cannot directly connect to external databases and query their tables. This is forbidden! SharePoint instead produces well-tested data entities and associations that describe the relationship between individual data entities based on the BDC model. Developers can custom those entities and associations, and even create new associations that are not in the original model.

As you can see, each BDC model represents a general view of an external system, its data entities and its associations that can be used in your SharePoint application because they are converted into familiar (external) content types and (external) lists. From development point of view, there is no difference between an external content type and an internal content type and developers do not need to worry about how data are either retrieved or

updated from the external system. This fits very well with the idea of Semantic Web. SharePoint guys realized that external and legacy systems underpin most business enterprises and designed this magnificent functionality very carefully. The brown field development was not so gloomy as we expected!!

4.3. Merging data in SharePoint

The equivalent SharePoint concept to merging data from multiple data sources in Semantic Web is to aggregate data from multiple lists into a single list. For example, in the company Contoso, a high-management member could have a request for gathering all expense reports belonging to accountability departments along with information related to orders from manufacturing department. All data could be stored in a single list to which the manager only has access. Once all data are brought together, we can develop any custom logic that implements any business logic, including Semantic Web inference patterns.

Wait a moment! A list can hold multiple Content types. Are you sure?

A key factor that makes merging data possible is that a list can contain multiple Content types, including external Content types. This feature provides you enormous flexibility since a list can contain large amount of data similar to a relational database table but also allows us to store different data type. Something that a relational data table cannot do. This supports our theory of that a SharePoint list is not a relational data table.

According to our description of Semantic Web architecture, this large list can be seen as a RDF Store that stores and retrieves from different data sources (internal and external lists) to merge data into a single large list. The RDF Query Engine using like SPARQL can be represented by the SharePoint Query Languages such as LINQ (Language Integrated Query) or CAML (Collaborative Application Markup Language) that are very similar to SQL query languages. In case we want a more sophisticated query language that allows us to use Semantic Web inference patterns, we could either extend LINQ or develop custom code on data stored on the list and store inference results in the own list. This approach would be quite similar to a JIT RDF Inference Query Engine.

SharePoint provides a number of mechanisms to solve merging data from multiple lists called: Aggregation List patterns. Getting a hang of how to aggregate data from multiple lists into a single list is essential for developers because it is a usual request from users.

4.3.1. List Aggregation Patterns

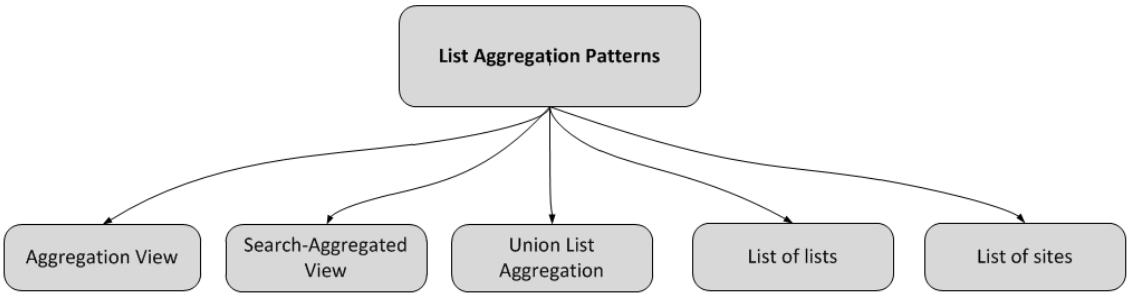


Figure 6.1 List aggregation patterns

4.3.1.1. Aggregated View

Description

An aggregate view uses the SharePoint APIs to query data from several data sources and aggregate it into a single view. This approach can return results from lists in the same Site collection.

Approach to implementation

1. Determine data sources from which we want to retrieve data
2. Determine what data you need to aggregate.
3. Create a custom Web page and use the SharePoint APIs to query the data.

Semantic Web scenario

This SharePoint scenario is slightly similar to Semantic Web scenario in which it is required to merge data from multiple webs hosted in a same organization. However, Aggregated View pattern does not seem to have a real application to Semantic Web scenarios since the scope of Semantic Web is much larger than few data sources hosted in a same organization.

4.3.1.2. Union List Aggregation

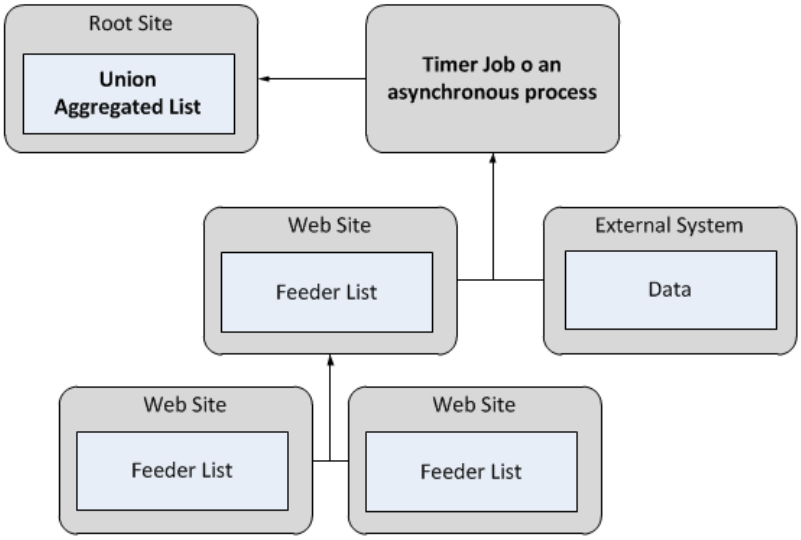


Figure 6.2 Union List aggregation pattern

Description

A union-aggregated list stores information from several lists or data sources. Usually this type of list is centrally accessible. These lists are easy to maintain because they allow users to manage information from many sources in a single location. Union-Aggregated lists contain data from data sources that share the same columns of data.

Approach to implementation

This type of list typically uses custom code to load the union-aggregated list with data. Timer jobs, asynchronous processes executed by SharePoint, usually perform this task.

Semantic Web scenario

This SharePoint scenario is quite a lot similar to Semantic Web scenario because in addition to an organization itself, this scenario can also involve merging multiple data sources beyond the organization's borders. Union List Aggregation is likely to be the best candidate to simulate a Semantic Web scenario since it allows you to work with internal and external systems. Remember "Green" and "Brown SharePoint data model scenarios.

4.3.1.3. Denormalized List Aggregation

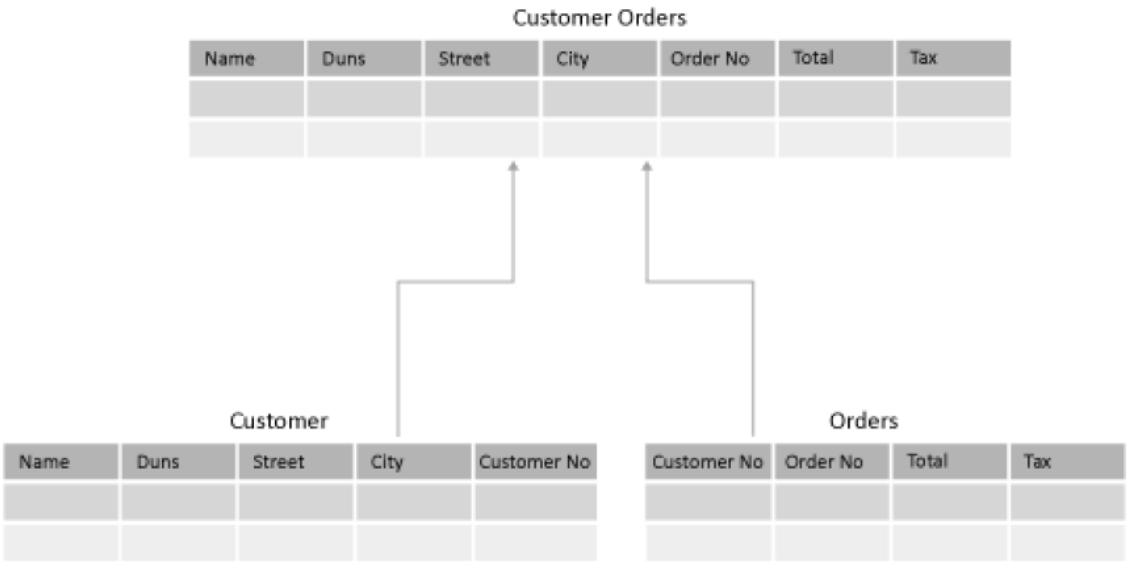


Figure 6.3 Denormalized List aggregation pattern

Description

A Denormalized aggregated list stores information from several lists or data sources using the same sort of process described in union-aggregated lists to perform the aggregation. Usually, this type of list is centrally accessible. These lists are easy to maintain because they allow users to manage information from many sources in a single location. Denormalized aggregated lists contain data from data sources whose columns differ.

Approaches to Implementation

The data is denormalized and the aggregated data contains different columns from several data sources. This approach uses custom code to load the denormalized aggregated list with data. Timer jobs or other asynchronous processes typically load this data.

Semantic Web scenario

It is difficult to find a Semantic Web scenario equivalent to this scenario because the resulting list is pretty similar to a relational database view rather than a typical Semantic Web representation of data. However, RDF inference patterns allows you to construct new data structure of data from other data such as set operations.

4.3.1.4. List of Lists

Description

A list of lists contains links to other lists. These lists are usually centrally accessible. Lists of lists appear in many different scenarios including lists of lists in the same site collection, multiple site collections, the same web application, multiple web applications, the same farm, and multiple farms.

Many times you will find that these lists are used to provide easy navigation to lists in many sites or across site collections, web applications, or SharePoint Server farms.

Approaches to Implementation

1. Determine the lists you want to provide links to.
2. Create the list to hold the links. (This is the list of lists)
3. Create the links to the other lists in the list of lists.

4.3.1.5. List of Sites

Description

A list of sites stores links to other SharePoint sites, or other web sites. These lists are usually centrally accessible. Lists of sites appear in many different scenarios including in lists of sites in the same site collection, multiple site collections, the same web application, multiple web applications, the same farm, and multiple farms.

Often you will find that these lists are used to provide easy navigation to sites across site collections, web applications, or SharePoint Server farms.

Approaches to Implementation

1. Determine the sites you want to provide links to.
2. Create the list to hold the links. (This is the list of sites.)
3. Create the links to the other sites in the list of sites.

Semantic Web scenario

Although both List of lists and List of sites patterns seems a bit dumb, they are actually tremendously effective and efficient. It might be because it is always easier to do than the way around but organizations almost always tend to think in both hierarchal and centralized terms. Either List of lists or List of sites patterns are painfully often found everywhere at times. The problem is that there is no equivalent scenario in Semantic Web.

4.3.2. Large lists Patterns

The challenge we face when we pull all data together into single lists is that as lists become larger, they can reduce the ability of SharePoint to operate efficiently and perform well. For example, viewing more than 2,000 items at a time from a list will impact performance, as will list queries that touch more than 5,000 items in the content database during execution. Performance will always benefit when we minimize the amount of list data retrieved, limiting it to only the data the user needs to perform his tasks. Large lists are not necessarily bad, and when properly managed, SharePoint can handle millions of items of data in a single list. However, large lists require proactive developer to ensure that they work smoothly on our Web site.

As you can see, both SharePoint large lists and RDF stores can suffer from the same problem with regard to performance. The great SharePoint large list's advantage over RDF store is that a SharePoint large list is unlikely to store as many data as a RDF store. The reason behind this assertion lies in the scope. The scope of a SharePoint application is typically an organization itself along with both its partners and customers, whereas the scope of Semantic Web Application can involve hundreds of organizations. It seems quite obvious that storage requirements in a SharePoint application are fewer than a Semantic Web one, thus the size a SharePoint large list with only a simple backup storage system like Microsoft SQL Server isn't a drawback.

SharePoint provides some patterns to deal with large list just in case and we will look through it just in case.

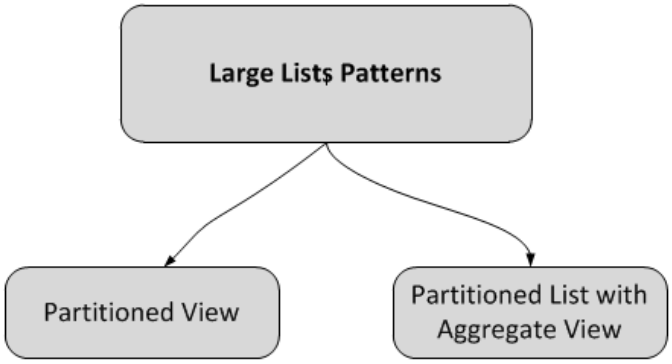


Figure 6.4 Large Lists patterns

Query Throttling and Indexing

In addition to large list patterns, SharePoint provides some features that allow us to keep an eye on performance degradation in large-size lists. On one hand, SharePoint enables you to restrict the number of items that can be accessed when you execute a query (by default, this limit is set to 5,000 items for users and 20,000 items for administrators). On other hand,

SharePoint enables you to index columns in a list. This is conceptually similar to indexing columns in a database table; however, in the case of SharePoint lists data, the index is maintained by SharePoint instead of databases.

4.3.2.1. Partitioned View

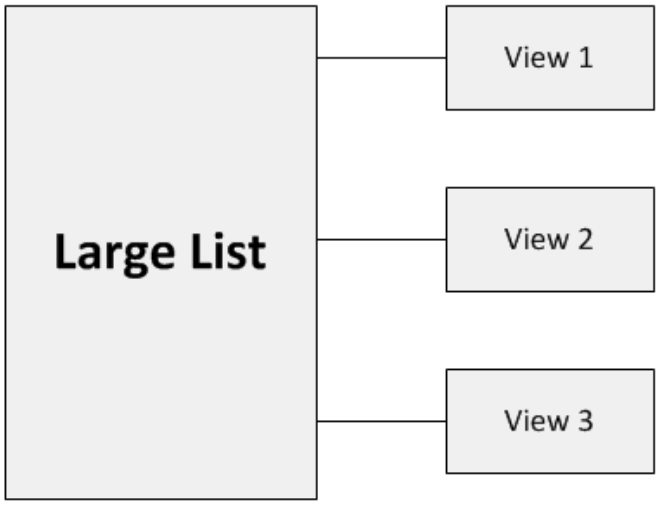


Figure 6.5 Partitioned view pattern

Description

View partitioning leaves the data in a single large list, but allows for access to the data in small segments through targeted views on a list. Often data can be segmented naturally—for example, by region, by status, by date range, or by department. This approach also efficiently supports multiple types of views on the same list because all data is in one place; thus, you could have a view by date range and by region for the same list. In order for partitioning to be effective, the columns being used to partition the view must be indexed.

Approach to Implementation

1. Remove default views assigned to lists (All list always has a default view).
2. Determine the column you want to partition on.
3. Create views for the partition.

4.3.2.2. Partitioned List with Aggregate View

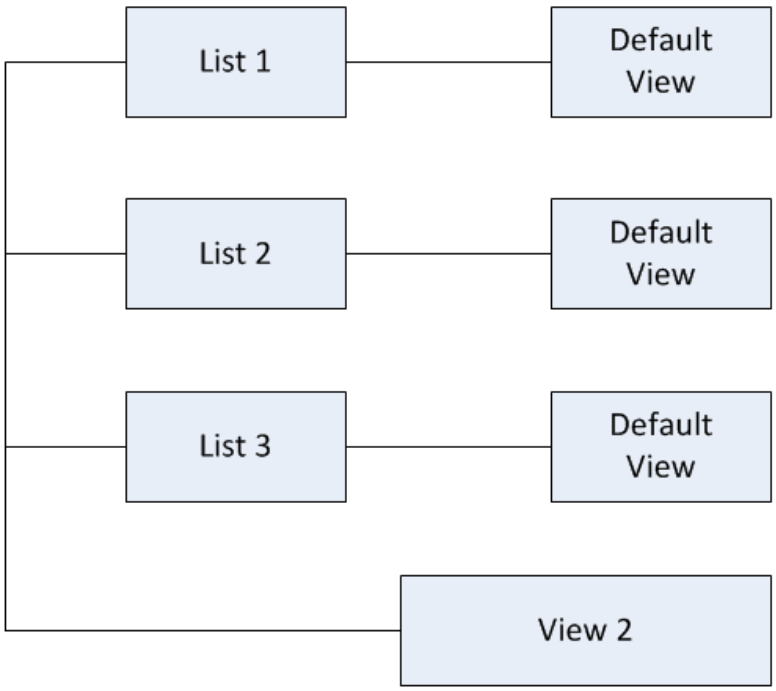


Figure 6.6 Partitioned List with view pattern

Description

Using the Partitioned List with Aggregate View pattern breaks the same type of data into individual lists. Typical usage of the list is through the default list views, but for specific cases items are aggregated across the lists into a central view. In this case you need to choose your segmentation strategy carefully because once you have segmented the data, segmenting a different column will require cross-list querying and filtering, which becomes more and more costly from a performance perspective as the number of lists grows. In order to do the aggregation you will need to define custom views to roll up data across lists. There needs to be a natural segmentation of data for this approach to work well. In order for partitioning to be effective, the columns being used to partition the view must be indexed; this will improve performance for aggregation.

Approach to Implementation

1. Determine the criteria you want to partition on.
2. Partition the data into separate lists based upon the criteria. All lists use the same content type.
3. Create aggregate views for the list.

4.4. Giving sense to data in SharePoint

4.4.1. Ontologies

SharePoint ships with excellent tools to combine data from multiple data sources as we have seen. Nonetheless, its lack of expressivity becomes apparent when we want to represent knowledge within a domain by defining a set of concepts and their relationships between them. This weakness lies in that SharePoint object model is built on Oriented-object principles and only allows us to describe basic notions of commonality and variability familiar from oriented-object languages — namely classes, subclasses, and properties. This is precisely the level of expressivity RDFS supports.

Should we want a higher level of expressivity like either RDFS-Plus or OWL offers, we need to develop custom code. But if so, we are no longer talking about a smart Semantic Web infrastructure but smart Web applications. We are getting back to the old-fashioned Web applications that are prone to inconsistent, disconnected and out-of-synchronization data. It is here the real issue. If the level of expressivity provided by RDFS satisfies our requirements, SharePoint is fine. But if we want higher level of expressivity, you may as well write custom .NET converters in such a way to convert SharePoint data mode into RDF data representation and to work on it.

Is it possible to build new inference patterns in SharePoint?

Apart from Content type inheritance, SharePoint unfortunately lacks more inference patterns, consequently building ontologies is quite limited. Unless you have battle-scarred developers work on extending the built-in SharePoint functionalities and features, implementing new inference patterns is almost a mission impossible. The problem is that all content types must internally inherit from the class Microsoft.SharePoint.SPContentType with no exception and this class is sealed. The same happens to the remainder of SharePoint data model classes such as SPField, SPList, SPListView, SPDocumentLibrary or SPListItem. The story is over.

Some inference patterns like those related to sets (union, interest, disjoint…) could be developed by custom code. Anyhow, we must be aware that those inference patterns will never be entirely integrated in the built-in SharePoint Object Model. The reason for this lack of flexibility is that SharePoint is very reluctant to permitting developers to develop solutions that extend SharePoint functionality and damage to its stability. SharePoint defines certain extension points that enable us to develop new functionalities but always under strict restrictions.

We will show a simple and typical scenario in which lack of inference can limit our options in SharePoint.

We have already seen in previous sections how to interpret information in a SharePoint list as RDF triples. Each list item in the list became a triple. The subject of the triple is the individual corresponding to the list item that the cell is in, the predicate is made up from the list name and the column name, and the list item's value is the cell contents. Turning to our Contoso company, now suppose that we have the product list that is stored in the Web site http://intranet.consoso.com/manufacturing within the Web application "Published Intranet Content" aimed for Manufacturing department. Table 7.1. Let's look at just the triples having to do with the Manufacture_Location.

```
mfg:Product1 mfg:Product_Manufacture_Location Sacramento
mfg:Product2 mfg:Product_Manufacture_Location Sacramento
mfg:Product3 mfg:Product_Manufacture_Location Sacramento
mfg:Product4 mfg:Product_Manufacture_Location Elizabeth
mfg:Product5 mfg:Product_Manufacture_Location Elizabeth
mfg:Product6 mfg:Product_Manufacture_Location Seoul
mfg:Product7 mfg:Product_Manufacture_Location Hong Kong
mfg:Product8 mfg:Product_Manufacture_Location Cleveland
mfg:Product9 mfg:Product_Manufacture_Location Cleveland
```

Table 7.1 Products SharePoint List in Manufacturing Web Site

ID (built-in)	Model Number	Division	Product Line	Manufacture Location	Available
1	ZX-3	Manufacturing	Paper Machine	Sacramento	23
2	ZX-3P	Manufacturing	Feedback Line	Sacramento	4
3	ZX-3S	Manufacturing	Sensor	Sacramento	34
4	B-1430	Control	Safety valve	Elizabeth	23
5	B-1430X	Engineering	Paper Machine	Elizabeth	14
6	B-1431	Engineering	Sensor	Seoul	0
7	DBB-12	Accessories	Monitor	Hong Kong	100
8	SP-1234	Safety	Safety valve	Cleveland	4
9	SPX-1234	Safety	Safety valve	Safety valve	14

Suppose that an external partner in the company keeps its own list of the products with information that is useful for that division's manufacturing activities— namely, it describes the sort of facility that is required to produce the part. This list is stored in the site http://partnerWeb.consoso.com/sites/products within the Web application "Partner Web" aimed for partners and its data comes from a legacy system like an ERP, therefore, an external list. We have already seen that integrating external databases and systems is a piece of cake in SharePoint and an external list behaves exactly as a normal SharePoint list does. Table 7.2 shows some products and the facilities they require. Some of the products in Table 7.2 also appeared in Table 7.1, and some did not. It is not uncommon for different lists to overlap in such an inexact way.

Table 7.2 Products SharePoint List in partner Web site		
ID (Built-in)	**Model Number**	**Facility**
1	B-1430	Assembly Center
2	B-1431	Assembly Center
3	M13-P	Assembly Center
4	ZX-3S	Assembly Center
5	ZX-3	Factory
6	TC-43	Factory
7	B-1430X	Machine Shop
8	P-1234	Machine Shop
9	1180-M	Machine Shop

If these two lists had been in the same Web Application, then there could have been a foreign-key reference from one list to the other by using a lookup column, and we could have joined these two lists together easily through lookup columns. Nevertheless, since the list come from two different Web applications, there is no such common reference.

When we turn both lists into triples, the individuals corresponding to each row list item assigned global identifiers. Suppose that we use the namespace p: for this second list. The triples corresponding to the required facilities are as follows:

```
p:Product1 p:Product_Facility "Assembly Center"
p:Product2 p:Product_Facility "Assembly Center"
p:Product3 p:Product_Facility "Assembly Center"
p:Product4 p:Product_Facility "Assembly Center"
p:Product5 p:Product_Facility "Factory"
p:Product6 p:Product_Facility "Factory"
p:Product7 p:Product_Facility "Machine Shop"
p:Product8 p:Product_Facility "Machine Shop"
p:Product9 p:Product_Facility "Machine Shop"
```

Although we have global identifiers for individuals in these tables, those identifiers are not the same. For instance, p:Product1 is the same as mfg:Product4 (both correspond to model number B-1430). How can we cross-reference from one list to the other? The answer is to use a series of owl:sameAs triples that allows us to establish that all statements about one instance hold for the other.

```
p:Product1 owl:sameAs mfg:Product4
p:Product2 owl:sameAs mfg:Product6
p:Product4 owl:sameAs mfg:Product3
p:Product5 owl:sameAs mfg:Product1
p:Product7 owl:sameAs mfg:Product5
p:Product8 owl:sameAs mfg:Product8
```

This solution has addressed the scenario for the particular data in the example, but the solution relied on the fact that we knew which product from one list matched with which product from another list. But owl:sameAs only solves part of the problem. In real data situations, in which the data in the list change frequently, it is not practical to assert the entire owl:sameAs triples by hand. In fact, the only sort of implementation we can hope in SharePoint is by custom code that can establish that two entities are equal: p.product1.Equals(mfg.product1). So, how can we infer the appropriate owl:sameAs triples from the data that have already been asserted?

By using RDFS, the right approach is to find an inverse functional property that is present in both data lists that we can use to bridge between them. When we examine Tables 7.1 and 7.2, we see that they both have a field called Model No, which refers to the identifying model number of the product. As is typical for such identifying numbers, if two products have the same model number, they are the same product. So we want to declare Model No to be an inverse functional property, thus:

```
mfg:Product_ModelNo rdf:type owl:InverseFunctionalProperty
```

This almost works, but there is still a catch: Each list has its own Model No property. There is another property, p:Product_ModelNo . So it seems that we still have more integration to do. Fortunately, we already have the tool we need to do this; we simply have to assert that these two properties are equivalent, thus:

```
p:Product_ModelNoowl:equivalentProperty
mfg:Product_ModelNo
```

Let's see how these inferences roll out. We begin with the asserted triples from both data sources and proceed with inferred triples. The last six triples are exactly the owl:sameAs triples that we needed.

```
p:Product1 p:Product_ModelNo "B-1430"
p:Product2 p:Product_ModelNo "B-1431"
p:Product3 p:Product_ModelNo "M13-P"
p:Product4 p:Product_ModelNo "ZX-3S"
p:Product5 p:Product_ModelNo "ZX-3"
p:Product6 p:Product_ModelNo "TC-43"
p:Product7 p:Product_ModelNo "B-1430X"
p:Product8 p:Product_ModelNo "SP-1234"
p:Product9 p:Product_ModelNo "1180-M"
mfg:Product1 mfg:Product_ModelNo "ZX-3"
```

```
mfg:Product2 mfg:Product_ModelNo "ZX-3P"
mfg:Product3 mfg:Product_ModelNo "ZX-3S"
mfg:Product4 mfg:Product_ModelNo "B-1430"
mfg:Product5 mfg:Product_ModelNo "B-1430X"
mfg:Product6 mfg:Product_ModelNo "B-1431"
mfg:Product7 mfg:Product_ModelNo "DBB-12"
mfg:Product8 mfg:Product_ModelNo "SP-1234"
mfg:Product9 mfg:Product_ModelNo "SPX-1234"
p:Product1 mfg:Product_ModelNo "B-1430"
p:Product2 mfg:Product_ModelNo "B-1431"
p:Product3 mfg:Product_ModelNo "M13-P"
p:Product4 mfg:Product_ModelNo "ZX-3S"
p:Product5 mfg:Product_ModelNo "ZX-3"
p:Product6 mfg:Product_ModelNo "TC-43"
p:Product7 mfg:Product_ModelNo "B-1430X"
p:Product8 mfg:Product_ModelNo "SP-1234"
p:Product9 mfg:Product_ModelNo "1180-M"
p:Product1 owl:sameAs mfg:Product4
p:Product2 owl:sameAs mfg:Product6
p:Product4 owl:sameAs mfg:Product3
p:Product5 owl:sameAs mfg:Product1
p:Product7 owl:sameAs mfg:Product5
p:Product8 owl:sameAs mfg:Product8
```

This sophisticated sort of inference patterns, typical though it is, is not possible in SharePoint unless we develop custom code. At this point, we can draw a very important conclusion on SharePoint inference patterns:

> "The level of expressivity in SharePoint is at RDFS level not higher"

4.4.2. Taxonomies and folksonomies in SharePoint

SharePoint ships with a service application that allow for the management of metadata across the entire organization called Enterprise Metadata Management. It makes a collection of metadata (columns) and content types available to all sites within a Server farm. Additionally, Enterprise Metadata Management allows us to define taxonomies and folksonomies, including synonyms or thesaurus.

Enterprise Metadata Management offers organizations some distinct advantages:

- **A global framework of content types and columns**. Organizations often want to guide all of their departments to use specific columns and content types, but also provide some flexibility so that each department can extend though inheritance to meet its specific needs.

- **Consistency in data entry**. It provides several key features that enable consistent entry of data. For instance, by configuring preferred values you can provide alternative selections to users when they are entering values. For example, if a user enters the term "car" or "vehicle," an alternate suggestion of "automobile" can be displayed for their selection. This feature provides a way to easily guide users to enter the preferred values for common words. In addition, as users are entering values for keywords, suggestions of previous keywords are displayed for selection.

Content types and columns for all lists within Server farm

In previous sections, we introduced that typically both content types and columns could be created and managed at Site Collection level, and consequently they were available to all sites within the Site collection. We lied to such an extent because SharePoint can alleviates this limitation by providing the ability to define content types in a central location called "Content type hub" and have them automatically published to numerous site collections. This feature provides a larger consistency in data in such way that a same content type can be used by distinct departments within an organization. For example, a document content type could be made of "Title", "Author" and "Description" columns and be made available to all department lists to make sure that everyone is using the concept of document properly.

Tagging content in SharePoint

In addition to content types and columns, Enterprise Metadata Management provides a way for organizations to bring certain order to the process of tagging content. Tagging content refers to applying metadata to documents and content within the organization. Tagging contents uses the concepts of managed metadata and keyword to refer to terms.

What are Managed Metadata and Managed Keyword in SharePoint?

Managed Metadata are actually special type of SharePoint columns. Managed Metadata can be assigned to content types as though they were normal columns. The most important difference is that they can be populated with some default terms in advanced in such way to guarantee that they are used consistently within the organization. Managed keywords are similar to managed metadata, with the key difference being that the end user is able to enter new terms. These are two extreme ends of the spectrum, and most organizations operate somewhere in the middle depending how tight governance has been set up in the organization.

Managed metadata in SharePoint are implemented through the use of a term store. When the Enterprise Metadata Management service application is provisioned, the term store is created. In order to organize terms, terms are grouped in term sets that allow us to structure them from broad to narrower

terms. At the lowest level are the terms, which are the values selected by users when they enter values in a column. Each of the terms can have an associated description, synonyms and even translation into other languages. Some typical examples of entries within a term store could include some of the following values: Departments, Offices, Categories, Project Status…

Simple Knowledge Organization System (SKOS)

In certain ways, Enterprise Metadata Management is quite similar to SKOS that is a family of formal languages designed for representation of thesauri, classification schemes, taxonomies, subject-heading systems, or any other type of structured controlled vocabulary. SKOS is built upon RDF and RDFS, and its main objective is to enable easy publication of controlled structured vocabularies for the Semantic Web.

4.5. Data access in SharePoint

SharePoint users can access to data easily from a user-friendly Web interface. But although SharePoint UI ships with SharePoint views that enable us to filter, sort and group data on SharePoint lists, it does not seem enough to get up to the level of queries languages like SPARQL.

In order to give support to querying data, SharePoint introduces several ways in which you can programmatically interact with your data. Most notably, LINQ to SharePoint allows you to build complex list queries with the user-friendly language integrated query (LINQ) syntax. LINQ even supports join predicates in queries, which moves the SharePoint list-based data model close to the power of a relational database.

Collaborative Application Markup Language (CAML)

SharePoint also allows us to construct queries with the cumbersome Collaborative Application Markup Language (CAML). CAML is a query XML-based language that is mainly used to define SharePoint data model building blocks like content types, lists, views… However, in spite of being tremendously prone to mistakes, sometimes developers have to resort to querying lists across Web sites by CMAL. It is undoubtly a large drawback because CAML has a bad name among developers.

SPARQL CONSTRUCT

As in SPARQL, LINQ has all the typical SQL operations but there is an essential difference in regard with SPARQL. SPARQL CONSTRUCT allows us to specify templates of new information based on patterns found in old information. A specification of this sort is sometimes called a rule, since provides a way to

specify things like “Whenever you see this, conclude that”. Examples of rules include data completeness rules (“If John’s father is Joe, then Joe’s son is John”),logical rules (“If Socrates is a man, all men are mortals, then Socrates is mortal”), as well as business rules (“Customers who have done more than 5000€ worth of business with us are preferred customers”)

Example of SPARQL CONSTRUCT

If we know how much business a customer has done with us, we can write a business rule in SPARQL to sort out our preferred customers.

:ACME :totalBusiness 5253.00
:PRIME :totalBusiness 12453.00
:ABC :totalBusiness 1545.00

The query

```
SELECT ?c a :PreferredCustomer .
WHERE { ?c :totalBusiness ?tb  .
        FILTER (?tb < 5000) .}
```

will assert all the preferred customers:

```
:ACME a :PreferredCustomers
:PRIME a :PreferredCustomers
```

If you might be wondering why SPARQL CONSTRUCT is so essential in SPARQL? Let us show you how inference triples are worked out from asserted triples and an inference pattern (also called rule) created by a SPARQL CONSTRUCT.

RDFS Domain inference pattern defined by SPARQL CONSTRUCT

By using SPARQL, you can define the inference rule for rdfs:domain

```
CONSTRUCT {?x rdf:type ?D } .
WHERE     {?P rdfs:domain ?D .
           ?x ?P ?y } .
```

If we have the following triples

```
:MarriedWoman rdfs:subClassOf :Woman
:hasMaidenName rdfs:domain :MarriedWoman
:Karen :hasMaindenName “Stephens”
```

from the domain inference rule, we can infer that

```
:Karen rdf:type :MarriedWoman
```

And from the inheritance rule, we can also infer that

```
:Karen rdf:type :Woman
```

LINQ vs. SPARQL

We already mentioned the lack of inference patterns in SharePoint and LINQ is claimed to have such problem as well. Nevertheless, SharePoint Object model plus LINQ has an advantage over SPARQL.

SharePoint ships with a modeling tool called SPMetal.EXE that enables us to automatically generate typed entities as an Object-Relational Mapping Framework. Those typed entities can be customized by developers by adding business logic. This feature overcomes the hardship of non-inference logic.

PART IV

“All great things are simple, and many can be expressed in single words.” —Winston Churchill

5. Conclusions

At the first part of the document, we hinted that knowledge management would require a certain rethink within the organizations. Given the increasing complexity and competitiveness, organizations drowned in mass of chaotic data were in need of pulling out more value from that data to convert them into true business assets. We concluded that this chaos of data was a logical result of the own human nature and it was worth accepting this scenario to deal with it rather than systematized and rigid environments.

Knowledge management is not a new challenge among organizations and some remarkable advances had already made. Knowledge management initially started off by a focus on content management and collaborative systems that sought to store and classify information into large data repositories. Nevertheless, from knowledge management's point of view, these efforts find themselves in a formative stage rather than a maturate stage since such systems rapidly scale back to "a place to lose information" in which data will wind up in inconsistent, disconnected and out-of-synchronization data. The platform Microsoft SharePoint is the one of the most foremost case, successful in collaboration but lacking in knowledge.

The advent of new technologies such as Semantic Web or Big Data, to help harvest, represent and distribute knowledge scattered throughout an organization, invites us to rethink knowledge management within organizations. What if Microsoft SharePoint and Semantic Web worked in tandem? How can this combination help us push knowledge management to have greater capabilities and benefit organizations?

At the second part of the document, we set off a journey through Semantic Web. We learnt that the essential notion of the Web is the idea of an open community in the Web: Anyone can say anything anytime (AAA Slogan). Consequently, billions and billions of unmanageable information out there lacking in consistency, connection and synchronization, as it were "dumb data". How to bring order into this chaos?

Semantic Web is based on the idea of a smart technology infrastructure that is capable of both bringing together all data from multiple distinct data sources and giving some order to this chaos of data. To achieve it, Semantic Web defines a set of standards and protocols called Semantic Web Technology Stack with the ability to work with raw data on which it builds up knowledge. Semantic Web Technology Stack is mainly made up of:

- a data representation language (RDF) to represent distributed data (triples)
- several domain modeling languages (RDF, RDFS, OWL) in the shape of ontology
- a novel query engine (SPARQL) with extra features added to the familiar SQL
- multiple computer-processable formats (N3, Turtle, XML/RDF)

An important stop on our journey through Semantic Web Technology Stack was the inference patterns or rules. With inference patterns, modeling languages reach higher level of expressivity beyond the familiar notions of commonality and variability typical of oriented-object languages. They can express detailed further constraints between classes, entities, and properties out of oriented-object languages' reach.

Finally, in our final stop on the journey, we put ourselves in the picture of a typical Semantic Web architecture to gain insight into how a Semantic Web application looks like and we introduced the concept of RDF store as a database with the additional ability to merge data from multiple data sources.

At the third part of the document, we got down to SharePoint. We learnt that at its core, SharePoint is a data provisioning engine and its fundamental design is based on the idea of using Web-based templates to create sites and lists to store and organize data.

- **SharePoint is designed on the concept of a server farm** that is composed of several Web applications hosting Web sites, service applications providing additional shared functionality, and relational databases whose access is forbidden.

- **SharePoint data model is conceptually quite similar to relational data model and, in turn, to RDF data model.** However, SharePoint does not have the same flair for establishing relationships between entities and properties as does RDF. A hardship that can be overcome by turning from SharePoint data into RDF data.

- **SharePoint data model is made up of columns (or fields), content type, list and list item.** Columns are metadata assignable to content type analogous to RDF properties. Content types are abstractions of domain comparable to RDF entities. List item are instances of content type corresponding to a set of RDF triples. Lists are the storage mechanism akin to RDF store.

- **SharePoint ships with excellent integration tools with existing and legacy systems** since it understands such systems are underpinning most business within organizations. SharePoint enables us to easily merge data from distinct data sources as does Semantic Web.

- **SharePoint concept equivalent to merging data from multiple data sources in Semantic Web is to aggregate data from multiple lists into a single list (RDF Store)**. SharePoint provides mechanisms to solve merging

data called Aggregation List patterns and means to handle large lists named as Large List patterns.

- **SharePoint gets up to the level of expressivity belonging to RDFS** except for property inheritance, that is to say, SharePoint allows us to describe basic notions of commonality and variability familiar from oriented-object languages — namely classes, subclasses, and properties. Its weakness lies in that SharePoint object model is built on Oriented-object framework.

- **SharePoint allow us to define taxonomies and folksonomies, even synonyms or thesaurus.** It stands for an excellent way for organizations to bring certain order to the process of tagging content.

- **SharePoint ships with LINQ, a query language to interact with data.** SharePoint views also that enable us to easily filter, sort and group data on lists. LINQ does not permit us to define inference rules but LINQ makes it up to by allowing customizable typed entities.

- **SharePoint is customizable and business logic can be easily developed** to work on SharePoint data model building blocks and compensate the lack of inference patterns.

SharePoint ships with excellent tools to integrate data from multiple data sources. However, its weakness becomes apparent when it is time SharePoint brought order to those chaotic data. SharePoint does not have the same level of expressivity as does Semantic Web. Establishing relationships between data is also far and away harder than Semantic Web. Nevertheless, as we mentioned throughout the document, the solution to this problem is almost always compensated by custom code. We consider that since we established certain foundations, a promising future project could entail developing custom converters with the ability to convert SharePoint data into RDF and extent the SharePoint Search to include inference patterns. This was beyond the scope of this project and it would be very interesting to go further.

As for project objectives, we think that Semantic Web is a state-of-art technology in a formation stage. Consequently, we found a great many hardships, particularly at the time we were looking into how to relate Semantic Web concepts to built-in SharePoint tools. However, despite some setbacks found, we consider the project objectives were met since we learnt Semantic Web principles, relation between Semantic Web and knowledge management, and the semantic side of SharePoint. We also could come to a very interesting conclusion: Has SharePoint become a place to lose data with semantic or not, it is mainly owing to the fact that it has not been customized properly. It seems to us that the problem does more with lack of management itself rather than a simple technological question. In other words, what data can benefit most my business? What data must be integrated? What

relationships must be established between data? This is more about management rather than technology. Knowledge raises from people, not technology.

6. Bibliography

[1] Microsoft MSDN. (2010). *Planning guide for server farms and environments for Microsoft SharePoint Server 2010*. Used to describe sections 4.2 "SharePoint Architecture" . Some figures were directly extracted from the book.

[2] Bill English. (2010). *Microsoft SharePoint 2010 Administrator's Companion*. Microsoft. Mainly used to describe sections 4.2 "SharePoint Architecture" .

[3] Todd Klindt. (2010). *Professional SharePoint 2010 Administration*. Wrox. Mainly used to describe sections 4.4 "Giving sense to data in SharePoint".

[4] Reza Alirezaei. (2010). *Professional SharePoint 2010 Development*. Wrox. Mainly used to describe sections 4.4 "Giving sense to data in SharePoint".

[5] Ted Pattison. (2011). *Inside Microsoft SharePoint 2010*. Microsoft. Mainly used to describe sections 4.2 "SharePoint Architecture" .

[6] Paolo Pialorsi. (2011). *Microsoft SharePoint 2010 Developer Reference*. Microsoft. Mainly used to describe sections 4.2 "SharePoint Data Model" and 4.5 "Data access in SharePoint".

[7] Microsoft Patterns and Practices. (2011). *Developing Applications Microsoft SharePoint 2010*. Mainly used to describe section 4.2 "SharePoint Data Model" " and 4.5 "Data access in SharePoint". Some figures were directly extracted from it.

[8] Dean Allemang. (2011). *Semantic Web for the Working Ontologist, Second Edition: Effective Modeling in RDFS and OWL*. Morgan Kauffman. Mainly used to describe section 3.2 "Semantic Web Technology Stack". Some excerpts were directly extracted from it.

[9] Toby Segaran. (2009). *Programming the Semantic Web*. O'Reilly. Mainly used to describe section 3.1 "Why do we need Semantic Web?" Some excerpts were directly extracted from it.

[10] *A story about Semantic Web*. (https://www.youtube.com/watch?v=bd8zR0v7Jts)

[11] *The Semantic Web* (https://www.youtube.com/watch?v=rhgUDGtT2EM)

[12] *The Semantic Wiki – Driving IT Organizational Clarity and Performance* (http://vaughanmerlyn.com/2012/02/21/the-semantic-wiki-driving-it-organizational-clarity-and-performance/)