UNIVERSITY OF OSLO
Department of Informatics

Towards an ontology for System Administration Case Study: Backup Operation

Karim Sani Ntieche
Oslo University College

May 23, 2007

Towards an ontology for System Administration
Case Study: Backup Operation

Karim Sani Ntieche
Oslo University College

May 23, 2007

Abstract

With the multiplicity of operating systems it is becoming common practice for organizations to deploy heterogeneous systems environments in order to benefit from their different advantages. The tradeoff of building heterogeneous environment is that it often leads to parallel support structures, non-interoperable management tools and system administrators with diverse skills to keep such complex infrastructures running. One basic requirement of interoperability or integration is the mapping between different models. This mapping can be carried out through syntactical and semantic translation using ontologies. This project focuses on the interoperability issues in heterogeneous environment, mainly mixed Unix/Linux and Windows infrastructures. The aim of this text is to investigate, with the help of a specific case study, how integration can be achieved in the management of Unix/Linux and Windows mixed environment through the knowledge sharing and interoperability capabilities provided by ontology engineering.

Acknowledgment

This thesis report is the conclusion of a challenging two years master's degree in network and system administration at Oslo University College in collaboration with Oslo University. There are several people i would like to thanks for their support and suggestions throughout this thesis work as well as the two years spent at Oslo University College.

First i would like to express my gratefulness to Oslo University College and Oslo University for giving me the opportunity to write this master thesis.

I would like to thank my thesis advisor Thor Hasle, whose advises have helped me keeping the right track in my work.

Special Thanks to Professor Mark Burgess for his dedication and enthusiasm throughout the degree and for his critics and suggestions during these last four months of thesis work.

Thanks to Joan Serrat and Martin Serrano of Polytechnic University of Catalonia for their hospitality and dedication during the four days training in Barcelona.

I'm grateful to all the other faculties at Oslo university College for their assistance during these two years. My appreciation to my fellow students for the supportive and friendly environment that has existed throughout this master program.

Last but not the least, many thanks to my family and friends whose encouraging words have helped when anxiety was ruling.

Special thought for my son Selim...

This work is supported by the EC IST-EMANICS Network of Excellence

Contents

1 Introduction **6**
- 1.1 Motivation . . . 6
- 1.2 Problem Description . . . 7
  - 1.2.1 Knowledge representation with Ontology . . . 8
  - 1.2.2 Ontology Mapping . . . 8
- 1.3 Research goals . . . 8
- 1.4 Outline of the remaining Chapters . . . 9

2 Background **11**
- 2.1 Current interoperability strategies in systems administration . . . 11
  - 2.1.1 Distributed Management Task Force(DMTF) Standards . . . 11
  - 2.1.2 CORBA . . . 16
  - 2.1.3 Vendor and Open source solutions . . . 16
- 2.2 Knowledge representation . . . 18
  - 2.2.1 Some basic definitions . . . 19
  - 2.2.2 Some Modeling languages for knowledge representation 20
- 2.3 Ontology for knowledge modeling . . . 23
  - 2.3.1 What is ontology . . . 23
  - 2.3.2 Use of Ontology . . . 24
  - 2.3.3 Typology of Ontology . . . 27
  - 2.3.4 Ontology representation languages . . . 28
  - 2.3.5 Ontology and Reasoning . . . 31
- 2.4 Ontology and interoperability: Related work . . . 32

3 Windows and Unix/Linux management: A short Comparison **35**
- 3.1 Comparing Windows and Unix/Linux systems . . . 35
  - 3.1.1 Flavors and versions . . . 36
  - 3.1.2 File Management . . . 36
  - 3.1.3 Security . . . 38
  - 3.1.4 GUI and command line interpreter . . . 39
  - 3.1.5 Management Cost . . . 39
- 3.2 Backup Operation Comparison . . . 40
  - 3.2.1 Windows Backup: . . . 41
  - 3.2.2 Linux Backup: . . . 42

4 Ontology design. Case Study: Backup operation in Windows and Linux **43**

4.1 Tools . . . 43
4.1.1 *Protégé* . . . 43
4.1.2 The Ontology mapping and merging tool: PROMPT . . 44
4.1.3 Reasoning tool: RACER . . . 45
4.2 Building the ontologies: The methodology . . . 46
4.2.1 Step One: Purpose and Scope . . . 47
4.2.2 Step Two: Considering re-using an existing Ontology . . 48
4.2.3 Step Three: Enumeration of important terms or concepts 48
4.2.4 Step Four: Defining Class and Class hierarchy . . . 49
4.2.5 Step Five: Defining properties of Classes . . . 52
4.2.6 Step Six: Defining and describing classes with properties restrictions . . . 55
4.2.7 Mapping the Ontologies . . . 58

5 Result Evaluations and Discussion 60
5.1 Ontologies evaluation . . . 60
5.1.1 Hidden assumptions: . . . 60
5.1.2 Quality of the knowledge capture and Usability . . . 61
5.1.3 Querying the ontologies . . . 62
5.1.4 Suggested framework for ontology evaluation . . . 63
5.2 Mapping evaluation . . . 63
5.2.1 Level of automation and accuracy . . . 63
5.2.2 Validation and Problems . . . 64
5.3 Discussion . . . 67

6 Conclusion and further work 69

A Appendices 74
A.1 Inferred taxonomy for the Windows *ntbackup* utility ontology generated in *Protégé* . . . 75
A.2 Inferred taxonomy for Linux *tar* program ontology generated in *Protégé* . . . 76
A.3 RDF/XML code generated with *Protégé* for the Windows *ntbackup* utility Ontology . . . 77
A.4 RDF/XML code generated with *Protégé* for the *tar* Linux backup ontology . . . 91

List of Figures

1.1 proposed architecture for mapping command line syntax between Windows and Linux 9

2.1 DMTF Technology Diagram[32] 12
2.2 WBEM Architecture [35] 15
2.3 CORBA Overall Architecture[39] 17
2.4 UML Class diagram 21
2.5 UML Use-case diagram 22
2.6 Usage of Ontology[1] 24
2.7 ontology as inter-lingua[1] 25
2.8 Illustration of the interoperability problem within and between the Fault, Configuration, Accounting, Performance, Security (FCAPS) functions based on the TMN model[2] 26
2.9 Categorization of Ontology as proposed by Mc Guinness et al in[22] 29
2.10 OWL level of expressiveness 31
2.11 Architecture of the management system approach using the Merge and Map method(M&M) to integrate diverse Network Management models[4] 34

3.1 Linux vs Windows Administrator productivity [38] 40

4.1 Property window in Protg-OWL editor 44
4.2 The PROMPT infrastructure and interactions between the tools. 45
4.3 Traversing the paths between anchors. The rectangles represent classes and labeled edges represent slots that relate classes to one another. The left part of the figure represents classes and slots from one ontology; the right part represents classes and slots from the other. Solid arrows connect pairs of anchors; dashed arrows connect pairs of related terms[15]. 46
4.4 Windows backup top level classes 50
4.5 Windows backup command line taxonomy for parameters . . . 51
4.6 Windows backup command line taxonomy for backup storage 51
4.7 Linux(tar) backup top-level classes hierarchy 52
4.8 Linux(tar) function and dataTobackup classes taxonomy 53
4.9 *notCompatibleWith* is a symmetric property between the two instances 54
4.10 *isSwitchOf* and *hasSwitch* are inverse properties 54

4.11 creating restriction with *Protégé*: here the universal restriction is applied to the class "dataStorage" 55
4.12 Diagram describing the Universal restriction (∀) 56
4.13 Meaning of the *hasValue* restriction(∋): Instances(specific files, tapes) from the "fileStorage" class are used with specific option switches(/F, /T, /N) in the "switch" class 58
4.14 mapping suggested automatically by PROMPT 59

5.1 Query result of parameters switches non compatible with the parameter switch /F in Windows 62
5.2 mapping suggested by PROMPT including some wrong suggestions such as mapping properties *hasName* and *hasSetTime* . . 64
5.3 psm mapping ontology class browser 65
5.4 Querying psm mapping ontology: Verification option(/V to -W) 66

Chapter 1

Introduction

1.1 Motivation

Organizations rely on their information system infrastructure to achieve their goals in an efficient manner. It can be recalled that 25 year ago this infrastructure was limited to huge computing machines kept in data centers and used only by specialized personnel. Now a days with the progress of the technology, computer usage has spread to all segments in most organizations hence becoming a working tool for most employees. This growth is continuously maintaining a challenge for system administrators who have the critical task of deploying and maintaining complex, heterogeneous systems. It has been argued in a recent research [30] that labor cost account for seventy percent of an enterprise's Information Technology(IT) cost. The heterogeneousness of computing infrastructure leads to the creation of separate support services which often use diverse terminologies while referring to the same administrative tasks. There are indeed several implementation differences between Windows and Unix/Linux systems, but also terminology difference for the same concepts. Because of this, specialized administrators often deal with each system within parallel support infrastructures. The poor communication between the two support structures might represent lost in productivity for the organization as they often appear to be antagonist entities with competing goals. Because of this fact it is legitimate that enterprises are interested in any technologies that can help reducing the cost of labor. One way to attack the cost of labor is to find ways to automate routine tasks through the development of sophisticated automated tools to manage systems while allowing system administrators to focus on other critical issues. Another way to reduce labor costs is to reduce management complexity by standardization which reduces management cost, but to be able to benefit from added power, performance and flexibility most organizations are opting for a mix of operating systems in their infrastructure mainly Windows and Unix/Linux.

Despite a higher cost in management, heterogeneous infrastructures appear to be a necessary choice for most organizations for a better productivity. There is therefore a need to reduce or eliminate conceptual and terminological dif-

ferences existing in different operating systems implementations in order to achieve a shared understanding through the definition of an unifying framework for describing different view points that will serve as a basis for:

- Communication between operators.
- Interoperability between systems.
- Re-usability, Reliability of knowledge

"Ontology" is the term used to refer to the shared understanding of some domain of interest which may be used as a unifying framework[1]. The term finds its origins in philosophy where it is defined as "the study of being or existence". In computer science or information management it is defined as a class model that represents a set of concepts within a domain and the relationships between those concepts. It has been used mainly in Artificial Intelligence, semantic web, software engineering and information architecture as a form of knowledge representation. Applied to System Administration of mixed environment the concept of ontology could contribute in defining an unifying framework for representing system management tasks of Windows and Unix/Linux systems.

Beyond the knowledge classification or representation issue, Ontology is also about describing a knowledge domain with reasoning and logic. Ontologies provide a semantic for describing relationship between concepts in the domain of interest. This semantic enable the Ontology representation to provide both a taxonomy and meaning of the domain that allow for a better identification of similarities and differences between different ontologies, through the ontology mapping process. Ontology mapping is about linking concepts across different ontologies to achieve semantic integration. Consequently ontology mapping is central to the process of integration in cross platform administration using the ontology concept.

1.2 Problem Description

This project is not considering the monitoring issues in heterogeneous environment. There exist several information models such as WBEM[1], SNMP-MIB[2] which have been developed with good capabilities for monitoring resources in devices but with often limited configuration capabilities. This work targeted describing a semantically rich shared knowledge representation which can be used in performing administrative tasks across multiple platforms(Windows and Unix/Linux in this case) through ontology mapping. The case study in this report was the "backup operation" for both Windows and Linux systems. The tasks were to capture the knowledge for that operation in both systems by looking at the commands syntax, restriction between the options and the different computers resources involved. After this representation was achieved

[1]web-based Enterprise Management
[2]Simple Network Management protocol- Management Information Base

the semantic and syntactic mapping of the two representation was be performed. Ontology knowledge modeling was used to perform to describe the different concepts.

1.2.1 Knowledge representation with Ontology

There are different types of knowledge representation techniques mainly coming from the field of Artificial Intelligence. A thorough understanding of different knowledge representations is a vital part of Artificial Intelligence, since the ease of solving a problem is almost completely determined by the way the problem is conceptualized and represented. The same is true for the task of communicating knowledge.[12]. Ontology as a knowledge representation technique presents different uses or roles and topologies. The challenges were to map the scenario of this work to the appropriate ontology type and to follow a suitable ontology creation process methodology to efficiently capture the knowledge to be represented. *Protégé* developed at Stanford University is the tools that was used to create the different ontologies. The Ontologies were represented in computer readable code using a RDF/XML[3] generated by *Protégé*.

1.2.2 Ontology Mapping

The ontology mapping is a critical part of the integration process. There are several algorithm and tools for performing ontology mapping that have been developed including PROMPT the build-in mapping tools included with *Protégé*. The mapping can be done automatically (through inference), semi-automatically or manually. The challenge to have an effective automatic mapping is to describe semantically enough the concepts in the ontology to be mapped. The mapping process evaluation was aimed to provide answers to the following question:

- How does this mapping contribute to achieve cross platform administration?
- Was PROMPT a suitable mapping tool to be used, is it suitable for system administration task?
- Were the ontologies semantically rich enough to facilitate the mapping process?

1.3 Research goals

The overall goal of the thesis was to be able to create semantically rich ontologies for the ”backup operation” for Windows and Linux which would describe as accurate as possible corresponding command line syntax in order to

[3]Resource Description Framework/Extensible Markup Language

facilitate the creation of Mapping definition between the two ontologies as illustrated in figure 1.1. The objectives of this thesis work were summarized as follow:

- To highlight some of the important differences and similarities between Windows and Unix/Linux systems such as Linux with respect to system administration related tasks. The focus will be aspects such as file permissions, services configuration, command lines and graphical interface and finally the "backup operation" which is the study case in this report.
- To create for the backup case study, Linux and Windows ontologies representing the command line syntax for each system using *Protégé* and ultimately perform mapping using PROMPT mapping tool automatically(preferred) or manually.
- The work aimed to suggest if ontology approach presents a valuable platform to achieve interoperability in system administration of heterogeneous systems.

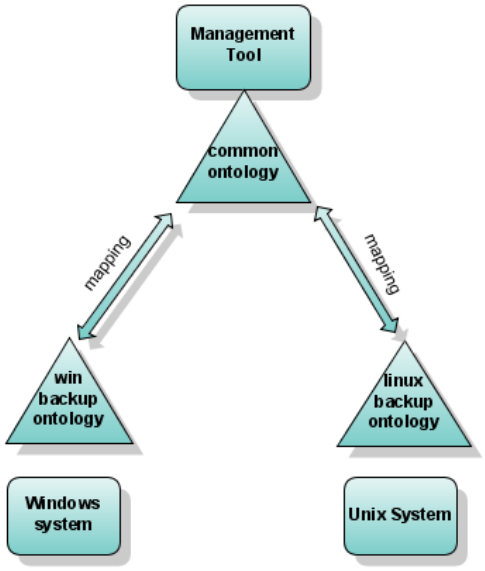

Figure 1.1: proposed architecture for mapping command line syntax between Windows and Linux

1.4 Outline of the remaining Chapters

Chapter 2 provides the reader with background information about current approaches in solving the interoperability problems in system administration such as standard information models and some proprietary solutions. The section also includes overview of the concept of Knowledge representation and

other knowledge representation techniques such as *Promise theory* and UML[4]. Finally it also provides an exhaustive summary on Ontology engineering.

Chapter 3 contains a short comparison between Windows and Unix/Linux system with an emphasis on the backcup task case study.

Chapter 4 describe the methodology used to create the different ontologies for the backup task in Windows and Linux as well the mapping process using the tool *Protégé*.

Chapter 5 presents the results evaluation and an overall discussion of the entire process.

Chapter 6 concludes the work with suggestions for further work.

[4]Unified Modeling Language

Chapter 2

Background

2.1 Current interoperability strategies in systems administration

When we look at the structure of organization now a days there are often a diversity of network devices, management tools, operating systems and applications that run concurrently to achieve the same goal of increasing the productivity of the organization. Cost of administration of systems as led to the development of automated tools. But as a complete automation of system administration tasks is yet to be achieved, specialized administrator are still require to perform critical system tasks. The Challenge for the system administrator is to master these different systems with their diverse configuration files and commands. An information model is an abstract but formal representation of entities including their properties, relationships and the operations that can be performed on them. There are several information models standards that have been created to allow different systems to share a common terminology in representing computer resources. This section presents some industry standards that are used to manage heterogeneous system with their limitations with respect to system administration tasks. Other vendors and open sources solutions to the interoperability problems between Windows and Unix/Linux systems are also discussed.

2.1.1 Distributed Management Task Force(DMTF) Standards

In heterogeneous environment in which multiple vendor solutions are the norm, interoperable standards enable the integration and flexibility that are key to controlling cost. The DMTF work group has developed various documents, guidelines and standards specifications for DMTF technologies. These technologies are designed to work together to address the industry's needs and requirements for interoperable distributed management. They also provide well-defined interfaces that build upon each other with the aim of delivering end-to-end management capabilities and interoperability. The interrelationships between the DMTF technologies shown in figure 2.1 delivers incremental value throughout the stack, building added value with each additional

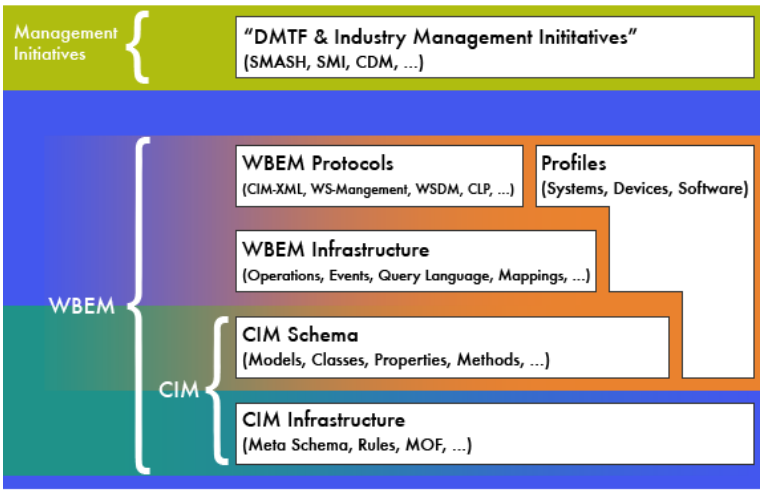

Figure 2.1: DMTF Technology Diagram[32]

layer that is implemented.

As the diagram2.1 shows, the foundation of the DMTFs technologies is the Common Information Model (CIM). The CIM Infrastructure specification defines CIMs "rules" and provides the details for integration with other management models. The next layer is the CIM Schema, which delivers semantically rich, object-oriented model descriptions for all managed elements. The CIM Schema facilitates streamlined integration and reduced costs by enabling the exchange of management information in a platform-independent and technology-neutral way[32].

Building upon CIM is the DMTFs Web-Based Enterprise Management (WBEM), a set of management and Internet standard technologies developed to unify the management of distributed computing environments. WBEM provides the ability for the industry to deliver a well-integrated set of standard-based management tools, facilitating the exchange of data across otherwise disparate technologies and platforms[32].

Also included in this diagram are Profiles, which provide a template to address specific management domains. By delivering a unified way to describe a given management domain in CIM, Profiles help with ease of use and offer a simplified means to achieve interoperable distributed management.
On top are the management initiatives from the DMTF, as well as other industry organizations that are built upon DMTF technologies. These initiatives, which deliver functionality to specific vertical applications and industries, include important implementations such as the DMTFs Systems Management Architecture for Server Hardware (SMASH) and Common Diagnostic Model (CDM), as well as the Storage Networking Industry Associations (SNIAs) Stor-

age Management Initiative Specification (SMI-S). These technologies from the DMTF deliver potent solutions, helping alleviate the challenges associated with managing todays complex, heterogeneous technology environments[32].

2.1.1.1 Common Information Model

A prerequisite of understanding and working with CIM is understanding object-oriented modeling. CIM is based on an object-oriented model. It is important to mention that object-oriented modeling is different from object-oriented programming. Object-oriented modeling is a formal way of representing something in the real world. It draws from traditional set theory and classification theory. Some basics to keep in mind in object-oriented modeling are that [33]:

- Instances are things.
- Properties are attributes.
- Relationships are pairs of attributes
- Classes are types of things.
- Subclasses are subtypes of things.

The Common Information Model (CIM) is an approach to the management of systems and networks that applies the basic structuring and conceptualization techniques of the object oriented paradigm. The approach uses a uniform modeling formalism that supports the cooperative development of an object-oriented schema. The Common Information Model (CIM) specification describes an object-oriented meta model based on the Unified Modeling Language (UML). This model includes expressions for common elements that must be clearly presented to management applications (for example, object classes, properties, methods and associations). The specification defines the syntax and rules. The specification defines the CIM meta schema, each of the meta schema elements, and the rules for each element. The specification also defines a CIM syntax language based on Interface Definition Language (IDL) called Managed Object Format (MOF). The specification also defines the CIM Naming mechanism. The CIM Specification does not describe specific CIM implementations, APIs, or communication protocols . The CIM Specification also does not include the core and common models. These models are separate from the CIM Specification and are produced independently of the specification. CIM provides a common definition of management information for systems, networks, applications and services, and allows for vendor extensions. CIMs common definitions enable vendors to exchange semantically rich management information between systems throughout the network. CIM is composed of a Specification and a Schema. The Schema provides the actual model descriptions, while the Specification defines the details for integration with other management models. Since CIM is based on an object oriented paradigm, these entities are described as objects. CIM is part of the

WBEM (Web-based Enterprise Management) initiative, which is being defined by major network vendors and managed by the DMTF. The CIM is composed of two parts: The Specification, which describes the language, naming, and the mapping to other management models; and the Schema, which is a formal definition of the model[33].

2.1.1.2 Web-Based Enterprise Management (WBEM)

WBEM is a set of systems management technologies developed to unify the management of distributed computing environments. The DMTF has developed a core set of standards that make up WBEM, which includes the Common Information Model (CIM), CIM-XML, CIM Query Language, WBEM Discovery using Service Location Protocol (SLP) and WBEM Universal Resource Identifier (URI) mapping. In addition, the DMTF has developed a WBEM Management Profile template, allowing for simplified profile development to deliver a complete, standalone definition for the management of a particular system, subsystem, service or other entity[34]. Figure 2.2 presents WBEM architecture. To understand this architecture, it's important to consider the components which lie between the operator trying to manage a device and the actual hardware and software of the device:

- A Management Interface: An operator would probably be presented with some form of graphical user interface (GUI), browser user interface (BUI), or command-line interface (CLI). The WBEM standard do provide specification for this interface. This makes one of the strengths of WBEM as the human interfaces can be changed transparently with respect to the rest of the system.

- Application program Interface(API): The GUI, BUI or CLI will interface with a WBEM client through a small set of Application Program Interfaces. This client will find the WBEM Server for the device being managed (typically on the device itself) and construct an XML message with the request.

- Client Protocol: The client will use the HTTP (or HTTPS) protocol to pass the request, encoding in CIM-XML, to the WBEM server

- WBEM Server: The WBEM server will decode the incoming request, perform the necessary authentication and authorization checks and then consult the previously-created model of the device being managed to see how the request should be handled. This model is what makes the architecture so powerful: it represents the pivot point of the transaction with the client simply interacting with the model and the model interacting with the real hardware or software. The model is written using the Common Information Model standard and the DMTF has published many models for commonly-managed devices and services: IP routers, Storage Servers, Desktop Computers, etc.

- A Provider: for most operations, the WBEM server determines from the model that it needs to communicate with the actual hardware or software. This is handled by ”providers”. Providers are small pieces of code that interface between the WBEM server and the real hardware or software.

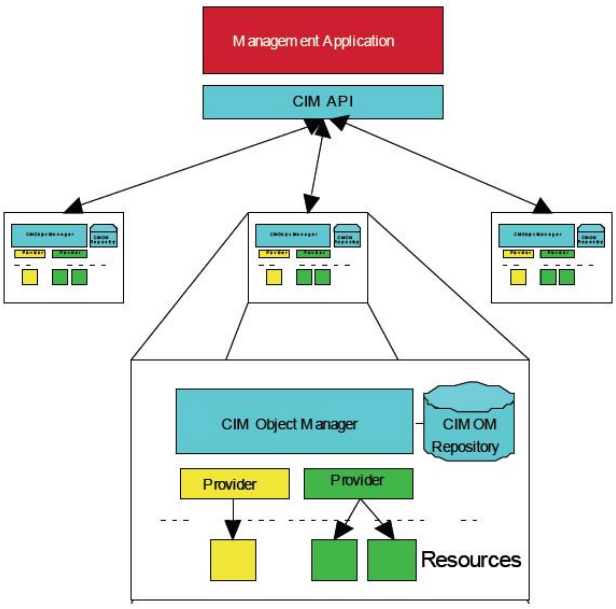

Figure 2.2: WBEM Architecture [35]

There exists several vendors implementations of WBEM

- Novell has adopted the OpenWBEM open source implementation of WBEM and includes it in SUSE Linux Enterprise Server
- Sun Microsystems includes its own Java WBEM Services in Solaris
- Microsoft has developed the WMI technology and has included it in Microsoft Windows
- RedHat has developed CimBiote[1]

These DMTF standards represent a good platform for describing resources (device, application, file, etc.) in a computing device. However they are mainly used for monitoring and can only provide limited modification capabilities. Monitoring represents only a portion of system administration. To achieve interoperability of system administration tasks which require modification of resources such as file permission, user management or operation such as backup, these standards are not appropriate as they are meant mainly for monitoring purpose.

[1] http://cimbiote.et.redhat.com/

2.1.2 CORBA

The Common Object Request Broker Architecture (CORBA)(figure 2.3) is structured to allow integration of a wide variety of object systems. The key to understanding the structure of the CORBA architecture is the reference model, which consists of the following components[37]:

- Object Request Broker(ORB): enables objects to transparently make and receive requests and responses in a distributed environment. It is the foundation for building applications from distributed objects and for interoperability between applications in heterogeneous and homogeneous environments.
- Object Services: a collection of services (interfaces and objects) that support basic functions for using and implementing objects. Services are necessary to construct any distributed application and are always independent of application domains. For example, the Life Cycle Service defines conventions for creating, deleting, copying, and moving objects; it does not dictate how the objects are implemented in an application. Specifications for Object Services are contained in *CORBAservices: Common Object Services Specification.*
- Common Facilities: a collection of services that many applications may share, but which are not as fundamental as the Object Services. For instance, a system management or electronic mail facility could be classified as a common facility. Information about Common Facilities will be contained in *CORBAfacilities: Common Facilities Architecture.*
- Application Objects: These are products of a single vendor on in-house development group that controls their interfaces. Application Objects correspond to the traditional notion of applications, so they are not standardized. Instead, Application Objects constitute the uppermost layer of the reference model. The Object Request Broker is the core of the reference model, combined with the Object Services, it ensures meaningful communication between CORBA-compliant applications.

There is a definition of an "object model", which defines what is the CORBA space. The object implementation provides the semantic of the objects. In this sense this object model can be consider as a step towards an ontology. The CORBA project also included notions of ontologies through a glossary of terms to be used in the object model. However the glossary is not itself an ontology but rather represent an informal framework for shared understanding[1].

2.1.3 Vendor and Open source solutions

Building a cross-platform management and automation environment can be quite complex because system calls and commands between operating environments differ as each operating environment uses different application program interfaces and libraries etc. There have been some effort from both the

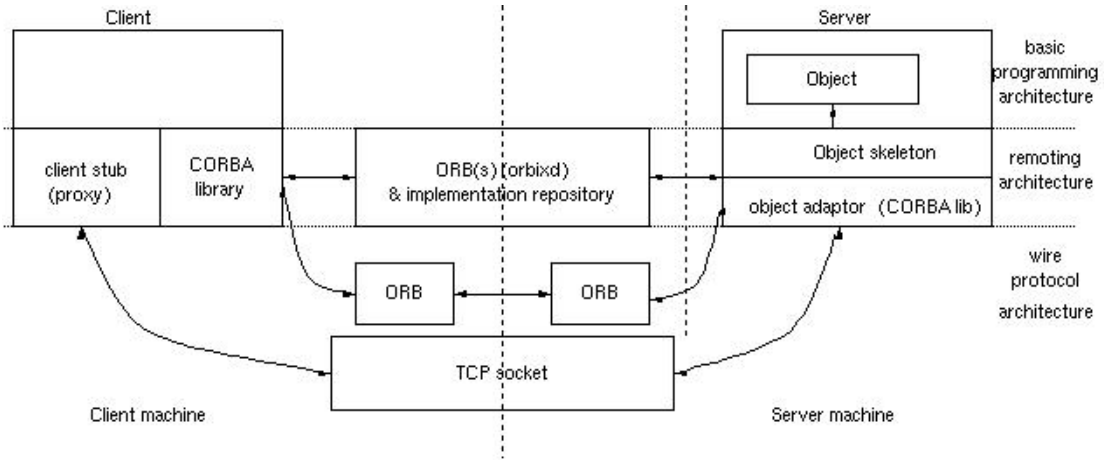

Figure 2.3: CORBA Overall Architecture[39]

vendor and open source communities to built set of tools, utilities, programs and libraries to achieve Unix/Linux and Windows integration.

2.1.3.1 An open source solution: *Cygwin*

cygwin is a collection of open source tools to allow various versions of Microsoft Windows to act similar to a Unix system. It aims mainly at porting software that runs on POSIX[2] systems (such as Linux, BSD, and Unix systems) to run on Windows with little more than a recompilations. It has of a library that implements the POSIX system call API in terms of Win32 system calls, a GNU development tool chain (such as GCC and GDB) to allow basic software development tasks, and a large number of application programs equivalent to common programs on the Unix system. Several Unix/Linux based applications such as Apache, X-Window, TeX, Gnome etc. have been ported to cygwin.

2.1.3.2 Some vendor solutions:Microsoft SFU/SUA and MKS Toolkit

Microsoft offers two UNIX/Windows integration products: Services for UNIX (SFU) and Subsystem for UNIX-based Applications (SUA). SUA being the new approach to the discontinued SFU. According to Microsoft, SUA is a "source-compatibility subsystem for compiling and running custom Unix-based applications on a computer running Windows server-class operating system". This subsystem is installed separately from Windows and operates as a guest UNIX operating environment on Windows systems operating as a POSIX UNIX environment[30].

MKS Toolkit for developers[3] includes a comprehensive UNIX/Linux and Windows integration management scheme designed to manage Windows environ-

[2]Portable Operating System Interface
[3]http://www.mkssoftware.com/

ments using UNIX commands and scripts. From a management perspective, the MKS Toolkit for Developers contains hundreds of authentic UNIX utilities such as *grep, df, du* and *ls*, as well as *ksh, csh* and bash shells enabling UNIX developers to immediately start using familiar scripts to manage UNIX as well as Windows environments. Password synchronization, remote utilities, and daemons are also supported. And UNIX commands can be used to perform automated back-ups across UNIX and Windows systems.

Microsoft with is SFU/SUA suite of products aims to control Unix environment from Windows while MKS Toolkits for developer emphasizes on the power of using the scripting expertise from UNIX developers to manage Windows and Unix in order to reduce the need of cross-platform training. Their goals are clearly antagonist as each one aim to have one vision (Unix or Windows) dominating the other.

2.1.3.3 Limitations

There is a certain bias either pro Windows or Unix in the way these tools are developed. Windows is trying to dominate the Unix world while the MKS toolkit is clearly more in the Unix side. Another issue about the scalability of these solutions arises if we think of more than just two different operating systems types that need to be integrated as most of these solutions require a lot of programming changes to the existing commands or programs to be able to execute in multiple platforms. This work investigates how knowledge sharing between system can help in translating commands between different system to achieve cross platform administration.

2.2 Knowledge representation

A thorough understanding of knowledge representation is important to computer science fields such as artificial intelligence, programming or system management. The ease of solving a problem or the task of communicating knowledge could be directly determined by the way the problem is conceptualized and represented. Several knowledge representation models have been developed to represent knowledge acquired from domain experts. However the concepts of model should not be confused with the one of an *architecture*. An architecture is a functional design while a model is an approximate representation of a system that makes a prediction. This infers the following requirements for a modeling language:

- The ability to organize information
- The ability to reason about information
- The ability to make predictions about behavior

2.2.1 Some basic definitions

It is necessary to define some basic concepts in order to make a clear distinction between concepts that seems similar while they actually have fairly different meanings[4].

- **Definition 1: Information**
  Information is defined by Shannon as a stream of symbols composed of some known alphabet. It can be quantified according to the basic results of information theory. Information is a very primitive or elemental concept. Although we sometimes use it in a high level sense, its precise meaning is at this low level. Information is essentially a form of coding.

- **Definition 2: Knowledge**
  Knowledge is the awareness and understanding of facts, concepts or information obtained by observing and reasoning about the world. It includes interpretations of facts that have been learned and reasoned about by an individual or entity.

- **Definition 3: Understanding**
  Can be defined as the construction of a model that incorporates the elements of knowledge within a subjectively consistent framework.

- **Definition 4: Model**
  A model is a collection of concepts, things (entities) and descriptions of their behaviors. It is any suitably idealized approximation to some phenomenon or system. A model is built on assumptions and leads to consequences or predictions.

- **Definition 5: Representation**
  A representation is an association or mapping between the actual elements of a model and some kind of descriptive medium that preserves (to some degree of approximation) the properties and relationships of the elements. Defining what is representation helps to identify the distinction between an ontology which is a shared understanding of a knowledge domain, and a representation of an ontology, which is an expression of the ontology in some kind of language. Languages like OWL[5], RDF[6] etc. are representations of ontologies, and in turn they can be expressed using representation such as XML[7].

[4] notes from a brainstorm session between Mark burgess, Thor Hasle, Demissie Aredo, Margareth Adaa and the author

[5] Web Ontology LanguageS

[6] Resource Description Framework

[7] The Extensible Markup Language

- **Definition 6: Architecture**
  An architecture is an explanation of structure, that is entities and their relationships. An architecture could be part of a model.

- **Definition 7: Specification**
  A general description of something that is made sufficiently specific; sufficiently usually implies the description will satisfy some constraints or requirements, or make some basic promises about its behavior

2.2.2 Some Modeling languages for knowledge representation

Two modeling techniques are discussed in this section; UML and Promise theory[8]. UML is a popular modeling language especially in the field of software engineering and there are currently several research works trying to use it as an ontology representation language. Promise theory on the other hand is a new modeling approach which aims at helping in the design of system management tools. This section also discusses the limitations of these modeling languages with respect to ontology representation.

Although these techniques have not been used in this work to represent the ontologies, it is important to mention that

2.2.2.1 UML

The Unified Modeling Language (UML) was created to be a specification language for programming, that is a way of representing requirements and tests in an abstract form. The meaning of modeling is somewhat restricted, though each revision adds new patches to extend its vision. The modeling facilities of UML include, among others, *classes* that can be used to represent the product's components (of any kind), *attributes* that describe properties of a class, *specialization relations* for modeling a taxonomic hierarchy of classes and *compositional relations* for modeling a partonomy[9](classification based on part-of relation) of classes. With these modeling facilities the product architecture can be specified[6]. UML models are represented diagrammatically. There are many categories of diagrams:

- Use-case diagrams: A use case is a set of scenarios that describing an interaction between a user and a system. A use case diagram displays the relationship among actors and use cases. The two main components of a use case diagram are use cases and actors. Figure 2.5 shows Use-case diagram.

- Class diagrams: Used to describe the types of objects in a system and their relationships. Figure 2.4 shows a class diagram.

[8]http://eternity.iu.hio.no/promises.php
[9]A classification based on similarities

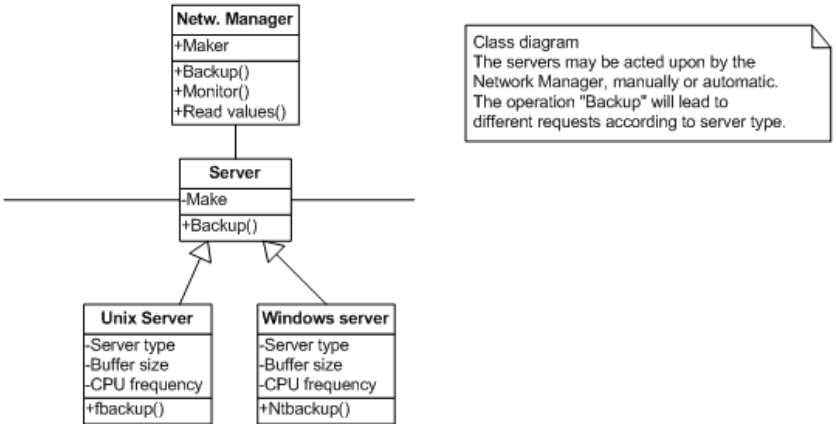

Figure 2.4: UML Class diagram

- Behavior diagrams(state chart, activity diagrams):These diagrams depict behavioral features of a system or business process. This includes activity, state machine, and use case diagrams as well as the four interaction diagrams.
- Interaction diagrams(sequence, collaboration): Interaction diagrams model the behavior of use cases by describing the way groups of objects interact to complete the task. The two kinds of interaction diagrams are sequence and collaboration diagrams.
- Implementation diagrams (component diagrams, deployment diagrams): The implementation diagrams are used in defining the requirements to deploy the system.

UML has a large community of experts users, but it is claimed in [13] that its lack of semantics to describe formally concepts in a domain and its limitation in reasoning capabilities do not allow it to be a prefer language for representing ontologies. As example, UML doesn't have the ability to represent the relationship "is similar to" , which is a critical relationship for heterogeneous end-to-end management, because it doesn't define logic mechanism to enable this comparison[26]. However several suggestions have been made to extend UML to allow it to represent ontologies. [23] investigated the use of UML and object constraint language(OCL) for the representation of information system ontologies. UML has not been used in this work to represent the ontologies.

2.2.2.2 Promise theory modeling

Promise theory was invented to discuss the issues surrounding autonomous operation, and voluntary cooperation. Unlike other modeling techniques, like Petri Nets or UML, promise theory is not about the stepwise development of a device. It is not about protocol modeling, rather it is about equilibria, that is

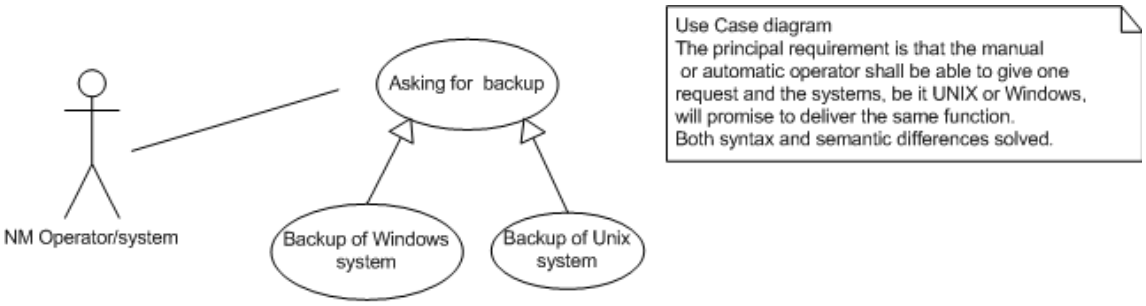

Figure 2.5: UML Use-case diagram

how to describe steady state behavior that has some underlying dynamics[9]. Promise theory is a model of advertised behavior. It deals explicitly with the advertisement of decisions that have been made.

Considering a number of agents, each with private knowledge. The agent's knowledge is ”flat”, it does not necessarily have a classification according to any particular model, but assuming that there exists a taxonomy of promise types that agents are assumed to agree on. Each agent has its own world view and only has access to information that it promised.

A promise model is a set of promises that will lead to interactions between the agents. The behavior of all agents might or might not be predictable from the promises made. An important question in promise theory is: can we predict how a collection of agents will behave?

Unlike algorithmic approaches to modeling, such as the many that are subsumed into UML, there are no sequential mechanisms in promise theory. It is, however, possible to promise ordered activities by introducing dependencies and conditionals. If one promise is conditional on another being fulfilled, then the actions which fulfill the promises must be ordered.

Facts are not explicitly represented in an ontology, but facts can be promised since they are simply a kind of knowledge or proposition. We cannot represent ”book X has been published” in the same way that one would in an ontology. We would have to introduce an agent, such as the publisher, who promised this knowledge, or even introduce the book as a ”dumb agent” who could promise this. This could seem artificial, but the great advantage is that there is no need to extend the theory to plural diagrams, as one would in UML. In this sense facts can be both promised to others and used in promise theory.

Promises focus on instances rather than generic classes. Thus there is never any doubt about where information is located: it always lies in the instance promising it. The typing of knowledge or information through promise types is sufficient to classify data in the sense of UML classes or ontological categories. This is simply a one-to-one mapping. One can, in principle, impose any

desired structure on promise types to reproduce programming data structures. Promise graphs can be used to reason in the sense that by following the chains of dependencies, one sees the functional processes that relate agents. One cannot take a specific fact that is not explicitly modeled however. Promises are often high-level things with the details of their implementation kept hidden.

Promise theory is a quite new initiative and because of the lack of existing tools to represent knowledge with this approach combined with the fact that it focuses more on instances rather than classes, it was not used in this work.

2.3 Ontology for knowledge modeling

2.3.1 What is ontology

Ontology has its origin in the field of philosophy where it refers to the study of existence. In computer science ontologies define theories of what exist. There are several ambiguous and similar definitions that have been given to the word ontology in different field of study such as Artificial Intelligence, software engineering, information system, knowledge engineering etc. In [11] Gruber defines Ontology as:

"An ontology is an explicit specification of a conceptualization"

This definition of ontology is said to create ambiguity mainly because of its brevity. The confusion of such definition might also be the fact that it uses terms that are already ambiguous and difficult to understand for someone new to the Ontology community. According to Gruber in [11] a conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose. Every knowledge base, knowledge-based system, or knowledge-level agent is committed to some conceptualization, explicitly or implicitly. Specification refers to definitions of classes, relations, functions, and other objects which make the Ontology.
John Strassner in [10] defines ontologies for network and system administration as:

An ontology is a formal, explicit specification of a shared, machine-readable vocabulary and meanings, in the form of various entities and relationships between them, to describe knowledge about the contents of one or more related subject domains throughout the life cycle of its existence. These entities and relationships are used to represent knowledge in the set of related subject domains. Formal refers to the fact that the ontology should be representable in a formal grammar. Explicit means that the entities and relationships used, and the constraints on their use, are precisely and unambiguously defined in a declarative language suitable for knowledge representation. Shared means that all users of an ontology will represent a concept using the same or equivalent set of entities and relationships. Subject domain refers to the content of the universe of discourse being represented by the ontology

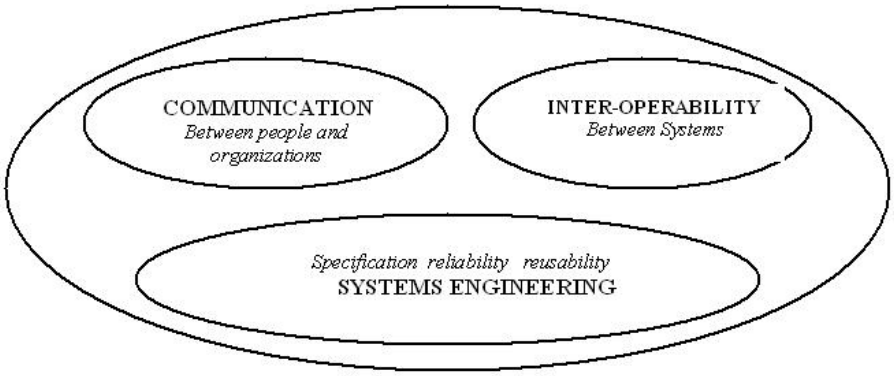

Figure 2.6: Usage of Ontology[1]

This is the definition that is assumed on this report as it suits the domain of system administration.

2.3.2 Use of Ontology

As it has been mentioned on the previous section, there are several descriptions and intended roles or usages for ontologies in different domains of application. This infers that the intended use of ontology might vary based on the problem. although, at a high level, most description seem to converge towards the role of re-usability of concepts. Some view their ontologies mainly as a mean to structure a knowledge base, others perceive it to be used as part of a knowledge base or just as an application-specific inter-lingua. Another important motivation for ontologies is to integrate models of different domains into a coherent framework; This is the case in business process engineering (where there is a need for integrated model of the entreprise's processes, organisations, goals and customers) or in distributed multi-agent architectures (where different agent needs to communicate and solve problems)[1]. The use of Ontologies can be classified into the following categories

- **Communication:**
  As stated earlier ontologies aim to reduce conceptual and terminological confusion by providing a unifying framework, thus enabling share understanding and communication between people. The shared understanding of a domain is important to communication between departments in an organization as well as for integration of multiple communicating agents with different perspectives. Ontologies aim also to provide consistency by reducing ambiguous definitions of terms in a domain.
- **Interoperability:**
  With respect to the interoperability issue, ontology is not about defining a single "uber-language[10]" that has no underlying business reason

[10] A common language to which all other languages can be translated to

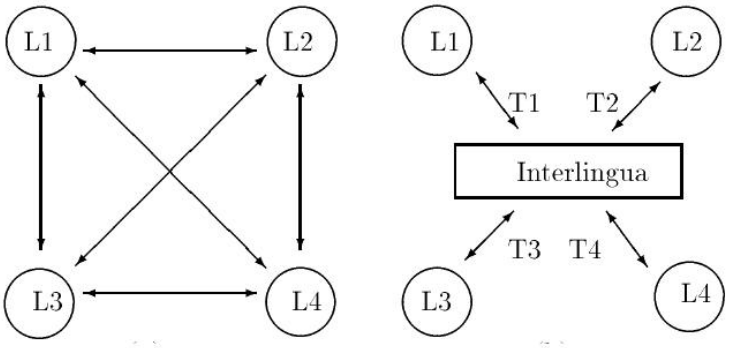

Figure 2.7: ontology as inter-lingua[1]

but instead achieve knowledge interoperability by using a set of ontologies to precisely and unambiguously identify syntactic and semantic area of interoperability between each vendor-specific language and programming model[13]. Several application of ontologies aim to address the issue of interoperability of software system, information model (CIM, CORBA)etc. For this role ontologies are referred as "Inter-Lingua" which assist to interoperability by supporting translation between different languages and representations as shown in figure2.7. This approach reduces the number of translator required for n languages(or representation) to n from n^2 where a unique translator is provided for a every two party.

The dimension of the interoperability needs to be specified. The dimension refers to the party involved in sharing the knowledge. in [1] the following categories are specified:

- Internal interoperability: Where the systems requiring sharing of knowledge are under the direct control of the same organization unit.
- External interoperability: This for the case of an organization that needs to insulate itself from changes made by a partner organization. It could also be the case for different departments within the same organization
- Integrated Ontologies among Domains: This is about integrating ontologies from different domains to support overall management in an organization. An organization might want to integrate knowledge from different layers(business, services, network etc.. within the same organization for a better work flow. Figure2.8 shows a

standard model with a number of management layers that help in managing the complexity of telecommunication network.

– Integrating Ontologies among tools: This will be to facilitate integration of legacy applications within the same domain.

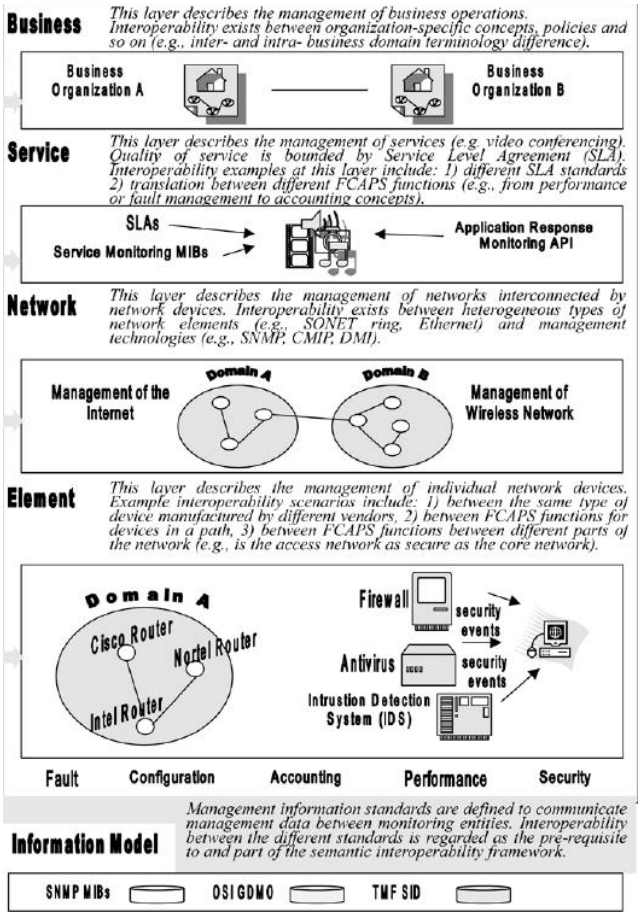

Figure 2.8: Illustration of the interoperability problem within and between the Fault, Configuration, Accounting, Performance, Security (FCAPS) functions based on the TMN model[2]

- **Systems Engineering:**
  This application of Ontologies support the design and development of software system by providing[1]:

  – Specification: A share understanding of the problem and the task at hand can assist in the specification of a software system. In an informal approach ontologies facilitates the process of identifying the requirements of the system and understanding the relationships among the components of the system. In an formal approach, an ontology provides a declarative specification of a software system which allows the users to reason about what the system is designed for, rather than how the system supports this functionality

- Reliability: Informally ontologies can improve the reliability of software systems by serving as a basis for manual checking of the design against the specification. Formally ontologies enables the use of semi automated consistency checking of the software system with respect to the declarative specification.
- Reusability: To be effective, ontologies must also support reusability, so that modules between different software systems can be imported and exported efficiently. When applying a software to a different domain from its original there is a risk of unexpected behavior. Ontologies provide a framework for determining which aspects of an ontology are reusable between different domains and task by characterizing classes of domains and tasks within these domains. Ontologies provide libraries of class objects for modeling problems that can be easily reused. The ultimate goal of this approach is the construction of libraries of ontologies that can be reused and adapted to different general classes of problems and environments.

2.3.3 Typology of Ontology

Ontology concept is an abstract concept. That might explain why there exist different proposed typology or categorization of ontologies. The type of the ontology to be created is related to the domain and intended usage.

In [16] four basic types of ontologies were proposed:

1. **Content Ontologies:** which also include

   - Domain ontologies: focuses on a particular set of related objects, activities or fields[13]. They are divided furthermore into:
     - Task dependent ontology: such an ontology do not require all the domain knowledge but some specific domain knowledge in a certain specific organization for a specific task.
     - Task-independent ontology: It's an ontology not related to a task but rather to an object or an activity . This type include:

       Activity-related ontology: This ontology is related to activities taking place in the domain and is designed having simulation of the domain activity in mind such as enterprise ontology. There are two major activities exist in a domain. One is behavior of an object and the other is organizational or human activities. Verbs play an important role in this ontology, however, they are different from those in task ontology. The subjects of the former verbs are objects, components, or agents involved in the activities of interest, while those of the latter are domain experts[19]. This type is again subdivided into *Object Ontology* which is about structure and behavior of an object and *activity ontology*.

Activity-independent ontology: such as *field ontology* which is related to the theories and principles ruling a domain. It include basic concepts of the theories, formulas, relations , and units involved in these theories.

- Task Ontology: Task ontology is a system of vocabulary for describing problem solving structure of all the existing tasks domain-independently. It does not cover the control structure but do cover components or primitives of unit inferences taking place during performing tasks. Task knowledge in turn specifies domain knowledge by giving roles to each objects and relations between them[13].
- General ontology: This type includes concepts not covered by the other domain ontology types

2. **Tell and Ask Ontologies:** This type focuses on sharing knowledge
3. **Indexing Ontologies:** These ontologies are specifically designed for querying.
4. **Meta-Ontology:** is defined in [16] as an ontology designed for representation knowledge. *Dublin Core*[11] is a metadata ontology that provides a vocabulary for describing the content of online information source

McGuinness et al. in [22] proposed instead a typology based on the richness of the ontology structure and the knowledge convey by the ontology. The figure 2.9 shows the suggested categorization that spans from, *controlled vocabularies* which is a finite list of terms (with no guarantee of uniqueness or unambiguity), to *advanced logical constraints* which include formal representation with object properties and restriction based on the specification of first order logic constraints between terms.

However as mentioned in [13], there are some other notable ontologies that are best classified as belonging to multiple groups of these schemes such as the *representational ontologies* than span multiple domains and provide representational entities without stating what should be represented. An example of this type of representational ontologies is the *Frame Ontology* which defines concepts such as frames, slots, and slot constraints, which enables other ontologies to be built using frame-based conventions. The choice of the type of ontology is a direct function of the requirements of the management information that needs to be represented[13].

2.3.4 Ontology representation languages

The potential of an ontology representation is closely related to the language used to represented it and the level of reasoning required. Reasoning requires

[11] http://www.dublincore.org/

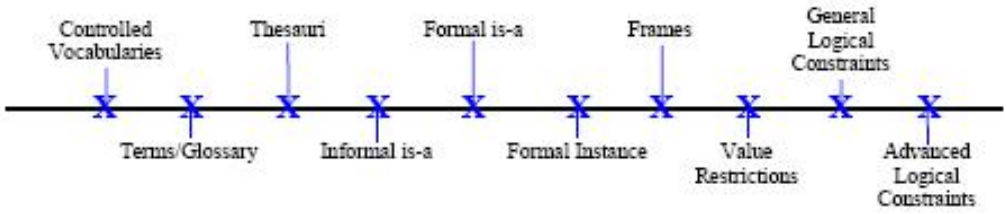

Figure 2.9: Categorization of Ontology as proposed by Mc Guinness et al in[22]

precision of meaning that is the reason why a preferred ontology representation language should include a formal mechanism for expressing semantics such description logics which can represent terms in a structured and formal way. The ontology representation language can be grouped as follow:

2.3.4.1 Logic-based Languages

These include:

1. Predicate Logic Approaches: Predicate logic approaches are based on first-order logic. This is a type of logic that extends propositional logic, whose formulae are propositional variable. The knowledge interchange format (KIF) is an important example of this approach. KIF provides a List processing-like syntax for expressing sentences of first order predicate logic and also provides extensions for representing definitions and meta knowledge. KIF is a highly expressive but low-level language for representing ontologies[23]. It is argued in [13] that this type of approach is suitable for experienced developer knowledgeable in logic programming, but more complex for general user.

2. Description Logic Approaches: Description logics (DL) model an application domain in terms of concepts (classes), roles (relations) and individuals (instances). The domain is a set of individuals; a concept is a description of a group of individuals that share common characteristics; roles model relationships between, or attributes of, individuals. Individuals can be asserted to be instances of particular concepts and pairs of individuals can be asserted to be instances of particular roles. LOOM[27] is a well known description logic language. It's a descendent of the KL-ONE family of DL languages. KL-ONE is a knowledge representation system in the tradition of semantic networks and frames language. KL-ONE implemented "structural inheritance networks": networks containing descriptions of named concepts with generalization/specialization links between them[23].

3. Frame and First-Order Logic: the frame based approach uses classes (or frames), some of which have properties called slots (or attributes). This approach has the following key elements concepts; instances, relations (which represent associations between different concepts), attributes (relation between a concept and a property of that concept), functions (a

special type of relation in which the last element of the relation is unique), and axioms (facts that are always assumed true, whether or not they can be formally defined by other components). *Ontolingua* and *FLogic*[12] are two examples of this type of representation language. FLogics is a formalism that accounts in declarative fashion for most of the structural aspects of object-oriented and frame-based languages. These features include object identity, complex objects, inheritance, polymorphic types, query methods, encapsulation and others[28]. Ontolingua although based on KIF aslo includes the frame ontology.

2.3.4.2 Markup ontology Languages:

These languages are most commonly XML based. Although XML has emerged in the Internet world as a standard representation format, which can he useful to describe and transmit management information, its formats alone do not give formal semantics to it[24]. Currently the biggest ontology driver is the Semantic Web and ontology languages increasingly rely on the World Wide Web Consortium (W3C) technologies [26]. Some of those technologies includes: The Resource Description Framework/Schema (RDF and RDFS), DAML+OIL[13] and the Web Ontology Language (OWL).

- RDF / RDF-Schema: RDF is a framework for metadata description. It employs the triplet model <object, attribute, value>, well-known in Artificial Intelligence community, in which object is called resource representing a web page. A triplet itself can be an object and a value. Value can take a string or resource. Object and value are considered as a node and attribute as a link between nodes. Thus, an RDF model forms a semantic network. RDF has an XML-based syntax(called serialization) which makes it resembles a common XML-based mark up language. In contrast with XML, RDF creates a new representation in which it contains meta information which usually do not appear in the original resource. Although RDF has been designed for metadata representation model, it can be used as a general-purpose knowledge representation, which might be apparent from the fact that it is a kind of semantic network model[29].
  The RDF-Schema (RDFS) as a semantic extension of RDF provides basic ontological modeling primitives, like classes, properties range and domains [26]. RDF Schema does not provide a vocabulary of application-specific classes and properties, but rather provides the facilities needed to describe such classes and properties. RDF schema has its built-in classes and meta-classes by which users can define any class and relation. *Rdfs:Resource* and its two subclasses: *rdfs:Class* and *rdfs:Property* are the key meta-classes. Every ordinary class defined in RDF Schema is an instance of rdfs:Class. In the same way, every property and relation defined in RDF Schema is an instance of *rdfs:Property*[29].

[12] Frame-logic
[13] DARPA Agent Markup Language + ontology integration language

- Web Ontology Language(OWL): is also a language developed by W3C. OWL is designed to make it a common language for ontology representation and is based on DAML+OIL. OWL is an extension of RDF Schema and also employs the triple model. Its design principle includes developing a standard language for ontology representation to enable semantic web, and hence extensibility, modifiability and interoperability are given the highest priority[29]. OWL has a layered structure which represent the different level of expressiveness as shown in figure 2.10: OWL Full (OWL DL syntax plus RDF), OWL DL (first-order logic only, roughly equivalent to DAML+OIL), and OWL Lite (a subset of OWLDL). OWL contains three types of objects: concepts, individuals, and properties. Since OWL is similar to DAM+OIL, inference engines used for DAML+OIL can also be used with OWL such as RACER[31].

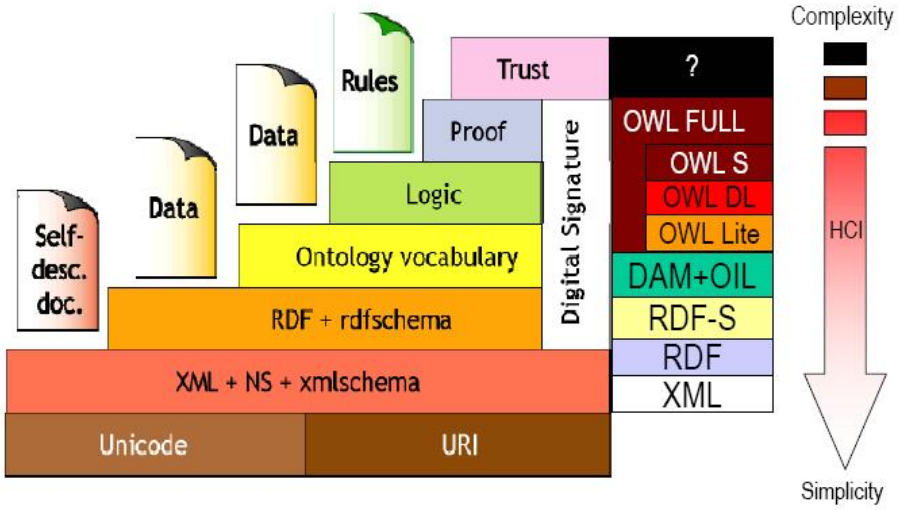

Figure 2.10: OWL level of expressiveness

2.3.5 Ontology and Reasoning

Reasoning means "to be able to deduce what must be true, given what is known"[13]. When choosing an ontology representation language, it is not sufficient only to consider the ease with which the language can be used to describe the domain. It is also necessary to consider the types of automated reasoning about ontologies that may be required. There is a well-known trade-off between the representational power of a formalism and the flexibility or cooperativeness of reasoning with it. As example, KIF provides all the expressive power of first order predicate logic, but reasoning about ontologies in plain KIF requires general theorem-proving capabilities. In contrast, description logic provides a much more structured and less general language for describing ontologies, and therefore specialised inferences can be performed on ontologies described using description logic. Much research has been undertaken to investigate the computational properties of various types of infer-

ences on different variants of description logic[23].

RACER(also called RacerPro) was the first OWL Reasoner. It has been continuously improved and currently still one of the fastest OWL reasoning systems available. With the exception of nominals, which are very hard to optimize, RACER supports the full OWL standard (indeed, nominals are supported with an approximation). *Protégé* supports an extended version of OWL (namely OWL with qualified cardinality restrictions) that is already supported by RACER with novel algorithms and optimization techniques. As most users use OWL for representing ontologies, in order to provide more flexibility than databases provide, with RACER, users can also describe their data and benefit from powerful ontology-based query answering systems. When RACER is used as a description logic reasoner, even more expressiveness than covered by OWL might be provided (e.g. constraint satisfaction, reasoning about topological relations, etc.)[31].

2.4 Ontology and interoperability: Related work

Different techniques have been used to resolve interoperability issues in diverse fields of study. Generally interoperability issues arise when different systems or knowledge representation want to exchange information. The terms *mapping* and *merging* are common while using ontology to solve interoperability problems. These techniques have been used together or separately to achieve integration of diverse systems. To avoid ambiguity there is a need to define those terms to distinguish their different roles:

- **What is Ontology mapping?:**
  It is the process of finding correspondences between the concepts of two ontologies. If two concepts correspond, they mean the same thing, or closely related things. The mappings should be expressed by some *mapping rules* which explain how those concepts correspond. The mappings are generated either by ontology experts or by some automatic tools.[5]

- **What is ontology merging?:**
  Ontology merging process is about identifying the similarities and differences between different ontologies with the goal of creating a single coherent ontology including terms from all merged ontologies.

- **What is ontology translation?:**
  Ontology translation is different from ontology mapping. The mapping is instead part of the translation process. Ontology translation is required to generate new ontology such as in cases where given *O1* and *O2*, two related ontologies and an extension *O1s* of *O1*, the translation process constructs the corresponding extension *O2s* of *O2*. This infers that ontology translation needs to know the mappings of two ontologies first, to accomplish its task[5].

There have been a lot of researches done mainly in the field of semantic web about ontology mapping. The interest in ontology mapping has spread to other domains such as biological and network management. In [3] Jorge et al. applied the ontology concept and principle to the definition and representation of management information. They suggested that when integrating management information models, the mapping and merging can be done with the help of ontology tools by creating a global management ontology with the associated mapping ontology. In [4] Jorge et a. proposed an ontology based method to merge and map network management models such as SNMP, DMI[14], CMIP[15] and CORBA. In that paper they proposed a technique for integrating information management model using a "merge and map"(M&M) method which includes a set of steps to help in th procurement of both the common model (through *merging*) and mapping rules. This M&M method is claimed to be more suitable for network management information than other proposed techniques which only deals with classes(properties not taken in account) or instances values(which are not known when merging information models)[4]. Figure 2.11 shows the management architecture proposed which basically consists of a common share ontology mapped to individual information model ontologies(gateways in the figure).

In [2] John Strassner et al. applied a similarity-based ontology mapping to solve the interoperability problem in router configuration management between Cisco and Nortel network devices. In their proposed method ontology mapping is done using the first order logic(FOL) calculus as the language for describing the semantics of the domain concepts and objects. Concept similarities was expressed through a function of logical weighted similarities. The weight of each aspect of a concept was assigned based on the application domain. This is important because similarity perception is related to the context or application domain. As example OSPF[16] and RIP[17] routing tables are similar regarding their structural aspect but dissimilar regarding their class reference(let say a classification based on routing algorithm).

[14]Desktop management Interface
[15]common information model protocol
[16]open shortest path first
[17]Routing information protocol

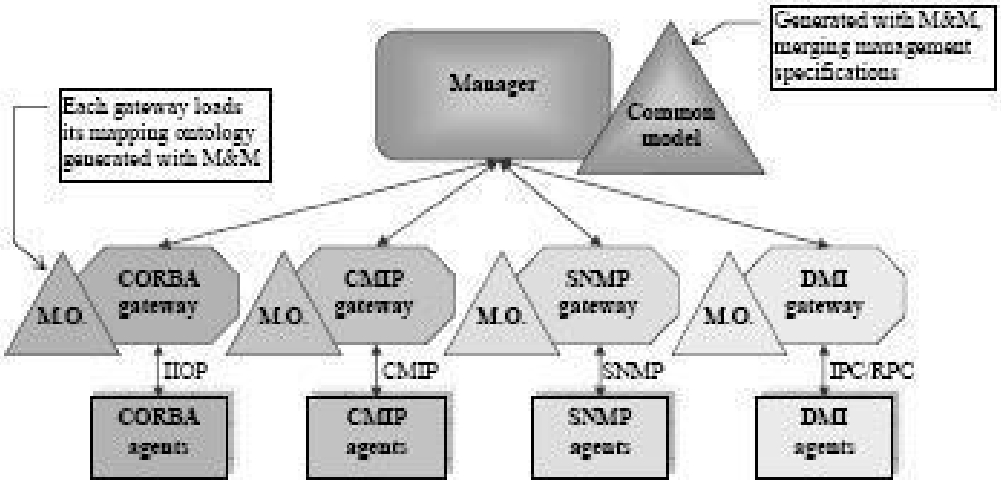

Figure 2.11: Architecture of the management system approach using the Merge and Map method(M&M) to integrate diverse Network Management models[4]

Chapter 3

Windows and Unix/Linux management: A short Comparison

This Chapter provides a short comparison between Windows and Unix-like system. Although there are references to earlier Windows version to NT the focus is the Windows NT family. Unix/Linux and Unix-like are used interchangeably to refer any Unix or Linux system. However the focus is on Linux operating system.

3.1 Comparing Windows and Unix/Linux systems

From normal users perspective Windows is referred as a user friendly operating system with higher cost than the unfriendly but ”free” Unix/Linux systems. To get beyond such superficial comparison it is important to look at the fundamental function of an operating system. An operating system (OS) is a computer program that manages the hardware and software resources of a computer. An operating system rationally processes electronic devices in response to approved commands. At the foundation of all system software, an operating system offers services such as:

- Memory management: controlling and allocating memory
- Process Management: prioritizing system requests
- I/O system Management: controlling input and output devices
- Communication: facilitating networking
- File Management: managing files
- Security
- Graphical User Interface / Command line interpreter.

There are difference in the implementation philosophy of these basics functions. Without discussing the details of the difference in the implementation of both operating systems. This section emphasizes on specific consequences of these implementation differences that are more relevant to this research and to system administrator in general:

3.1.1 Flavors and versions

Both Windows and Unix/Linux systems come in many flavors or versions. All the flavors of Windows come from Microsoft, the various flavors of Unix/Linux systems come from different sources; BSD[1], Linux (Linspire, Red Hat, SuSE, Ubuntu, Mandriva, Knoppix, Slackware, Lycoris), Sun Solarix, HPUX, AIX. Windows has two main lines: "Win9x", which consists of Windows 95, 98, Millennium, and "NT class" which consists of Windows NT, 2000,XP and now Vista. Currently Microsoft does not longer supports Windows NT and all the 9x versions.

In Linux, flavors are referred to as "distributions" . All the Linux distributions released around the same time frame will usually share the same kernel. They differ in the additional software provided, Graphical interface, install process, price, documentation and technical support which are the specificities of each distribution. Both Linux and Windows come in desktop and server editions. Linux appear to be more customizable than Windows. There are many special purpose versions of Linux above and beyond the full blown distributions described above. An example of special purpose version is NastLite[2]. NASLite is a version of Linux that runs off a single floppy disk and converts an old computer into a file server. This ultra small edition of Linux is capable of networking, file sharing and being a web server.

3.1.2 File Management

3.1.2.1 Disk file system structure

A file system is a method for storing and organizing computer files and the data they contain to make it easy to find and access them. File systems may use a storage device (disk file system) such as a hard disk or CD-ROM and involve maintaining the physical location of the files, they might provide access to data on a file server by acting as clients for a network protocol. Windows systems have moved from the less secure FAT(file allocation table) and FAT32 file systems to the NTFS(NT File system). Unix-like system have various file system such as ,XFS(X font server) UFS(Unix File System), ext2/3(Extended File System). File names are associated to an index in the a file allocation table in Windows or to an inode in Unix-like system.

[1] Berkeley Software Distribution
[2] http://www.serverelements.com/

A user shifting from Windows system can find confusing the directory structure under Unix/Linux system. Unix/Linux operating systems assign a device name to each device, but this is not how the files on that device are accessed. Instead, Unix creates a virtual file system, which makes all the files on all the devices appear to exist under one hierarchy. This means, in Unix, there is one root directory, and every file existing on the system is located under it somewhere. Furthermore, the Unix root directory represented by "/"does not have to be in any physical place. The root directory does not have to be on the local hard drive, Unix/Linux System can use a network shared resource as its root directory. Windows has various partitions and then directories under those partitions. These various partitions are detected at boot and assigned a drive letter. Under Unix/Linux systems, unless a partition or a device is specifically mounted, the system does not know of the existence of that partition or device. This might not seem to be the easiest way to provide access to a partitions or devices but it offers great flexibility. As an example one could move a the system executable available in the directory */usr* to another disk or network location without it affecting the system while on Windows it will be impossible to do the same operation with the directory C:\windows\system32 (or C:\winnt\system32) without getting errors messages.

3.1.2.2 Configuration files

One of the basic differences between Windows and Unix/Linux systems is the fact that UNIX/Linux configuration is defined in plain text files while Windows stores most of this information in binary format in the registry. The Windows registry provides centralized storage for information and settings about the hardware, operating system, applications, users, and user preferences on a Windows systems. The registry provides a hierarchical structure for settings, allowing keys to have subkeys and named values, similar to the directory and file structure of a file system[8]. It can be accessed and updated under software control and also directly by users. The registry first appeared in Windows 3.1. In that system it was a single file, called REG.DAT, and was mainly used to store information about OLE objects. Most other configuration data was held in various INI files, of which WIN.INI and SYSTEM.INI were the most important. The modern registry, as found in Windows 9x and NT family, brings together all the information that was previously held in REG.DAT and the separate INI files[7]. Although the registry is usually considered to be a single entity, its contents are in fact stored in more than one physical file. In Windows, the registry is spread over a series of files, sometimes called hives. SYSTEM.DAT and USER.DAT are usually held in the Windows directory. However, it is also possible to place USER.DAT in the users login directory on a network, thus allowing the user to log in at other workstations. In Windows, the hive files are located in the SYSTEM32\CONFIG directory.

3.1.3 Security

- **Authentication:**
  Windows NT relied on NTLM (NT lan manager) user authentication to store password in the security account manager(SAM). Two hashed versions of the password are stored, LM-hash[3] and NT-native, unless the system is told to just use one. The NT-native variant is stored using MD4 and the LM-hash using a variant of DES. From Windows 2000 Kerberos was included as the default authentication protocol. The Kerberos protocol is composed of three subprotocols. The subprotocol in which the KDC (Key Distribution Center) gives the client a logon session key and a TGT (Ticket Granting Ticket) is called the Authentication Service (AS) Exchange. The subprotocol in which the KDC distributes a service session key and a ticket for the service is called the Ticket-Granting Service (TGS) Exchange. The subprotocol in which the client pre-sends the ticket for admission to a service is called the Client/server (CS) Exchange[36].
  UNIX/Linux systems request a password to authenticate users identity. When a user has entered the password, it is encrypted and compared against the encrypted password stored in */etc/passwd* (or the NIS[4] database). If the two match, the user has proven to be a legitimate user in the system.

- **File security: Access Control**
  In Windows, object in the system has an Access Control List (ACL) associated with it. This list consists of a number of Access Control Entries (ACE). Every ACE is associated with a user (or a group) security identifier (SID) and holds the actions that this user is allowed or disallowed to perform on this object. ACEs that disallow are put before ACEs that allow in the ACL. A user that does not have an ACE in the ACL has no access at all to that object.When a user is authenticated to the system a token is created for this user. This token is called the primary token. It contains, among other things, the SID for the user and the SIDs of the groups that this user is a member of. This token is compared with an objects ACL to grant (or deny) the user access to this object[25].
  In Unix/Linux systems, access control is implemented through the file system. Each file (or directory) has a number of attributes, including a filename, permission bits, a user ID(UID) and a group ID(GID). The UID of a file specifies its owner. The permission bits are used to specify permissions to read (r), write (w) and execute(x) the file for the user, for the members of the user's group, and for all other users in the system. The permissions: *rwxr-x–x* specify that the owner may read, write and execute the file, while the group members are allowed to read and execute it, while all others only may execute the file. A dash (”-”) in the permission set indicates that the access rights are disallowed. Currently many Unix/Linux systems also support some form of ACL schemes[25].

[3] LAN Manager-hash
[4] Network Information Service

- **Viruses:**
  There are continuously several viruses and spywares created for Micorsoft Windows and many of these viruses have been successfully spread around the Internet with costly damage to organizations. It is sometimes argued that Unix/Linux system enjoy less viruses attacks because hackers are targeting only the system that dominate market share which is Windows. However the fact that the Apache webserver which accounts for more market share than the Microsoft Internet Information Service(IIS), has been less vulunarable to virus than the later is discrediting that claim. This exposure to viruses and spywares can only increase the cost of management.

3.1.4 GUI and command line interpreter

One of the most interesting feature that made Windows popular from its early days is the graphical user interface. This GUI has was the important contribution which made computers accessible to non-experts users. Windows GUI is continuously improving and there is definitely a huge difference from the initial GUI in version 3.1 compare to the current GUI in vista. Most of the configuration tasks are still performed through the GUI. Critics suggest that because of this constraint Windows administration is less flexible than Unix/Linux which relies much more on the command line interface and scripting power for system administration. However Windows has lately improved its command line facility with a new scripting engine called Powershell(realeased in novemver 2006). PowerShell requires version 2.0 of the Microsoft .NET Framework and runs only under Windows XP, Vista and Server 2003. Microsoft claims that that Powershell command line and scripting language offers as much flexibility and efficiency enjoyed in Unix/Linux systems.
On the other hand Linux systems have improved their GUIs and as a result there is an increase in the number of Linux desktop users. The most common Linux GUIs include: K Desktop environment(KDE) and GNOME.

3.1.5 Management Cost

The study of the total cost of ownership (TCO) of Unix/Linux and Windows has often being contradictory as some of these studies appeared to be bias on either side. Microsoft has released TCO studies which concluded that Linux had a higher TCO, predominantly due to higher management costs. This contradict claims made open source distributors such as RedHat or Suse. A recent survey [38] investigated whether Linux server management was a significant barrier to cost-effective Linux operation by analyzing the current state of Linux management and its associated cost. This study included important aspect such as provisioning, reliability, management(patch, security etc.), support, resource costs(Administrator salaries, training etc..). This study claimed to have found at worst a marginal difference in base resource costs between Linux and Windows. However, Linux resourcing becomes significantly less expensive when taking into account the ability of Linux to support larger numbers

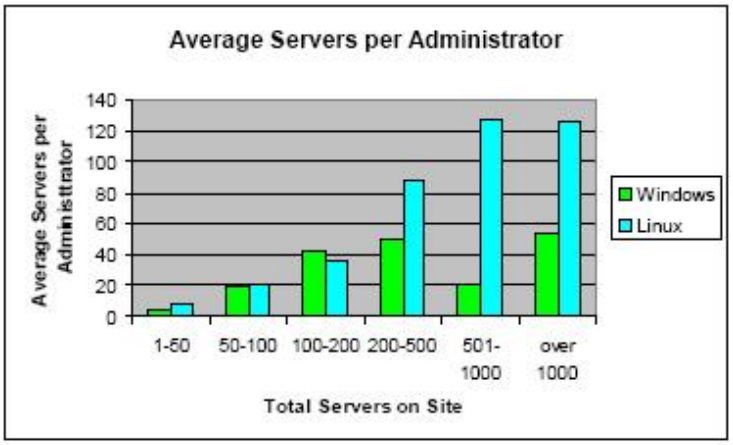

Figure 3.1: Linux vs Windows Administrator productivity [38]

of users, and the additional productivity of Linux administrators. Linux resources are easy to find, and tend to be highly experienced. Overall, resource costs for Linux environments are therefore likely to be lower than for Windows. In addition, administrators who can manage both Windows and Linux command around the same salary as Linux-only administrators, bring added efficiency to mixed environments. As shown in figure 3.1 one of the more interesting findings in that research is that Linux administrators tend to manage more servers than Windows administrators. From their results, in sites with up to 100 servers, each Linux administrator managed on average 15 servers, and each Windows administrator only 12 servers. This is in the opinion of this survey a balance to the salaries differences between Linux and Windows administrators. The survey suggested that the difference in acquisition costs for a typical Linux-based web application stack, including both hardware and software costs, is over ten times less than for an equivalent Windows-based environment to illustrate that the acquisition costs of Linux are lower. Admitting that the study was not exhaustive, it suggested that the choice of platform must account for many more variables than just resource costs, management effort, or even TCO. Windows in particular has many available and proven applications and has made good inroads on UNIX/Linux on the server side while the average resource costs for Linux are no longer significantly higher than for Windows[38].

3.2 Backup Operation Comparison

For the case study was started with a small requirement of backup in an organization. The scenario assumed an heterogeneous environment running Windows and Linux operating systems. One of the primary responsibility of a system administrator to plan and implement an efficient strategy to periodically copy files on the system to a remote location. Windows backup and Unix (linux) backup present divergence both in their implementation and syntax.

3.2.1 Windows Backup:

Windows Operating system uses *ntbackup* program to perform backup at the command line. It's important to specify that this command as evolved with the different version of Windows operating system. In a Windows system, the registry is very system specific and configurations and software installations are not simply a matter of dropping files on a system. Therefore to be able to restore such data there is a need of a specific tool such as the *ntbackup* utility. Windows NTFS file system has an archive bit for each file that can determine if the file was recently changed. This feature make it possible to perform several types of backup under Windows:

- Copy backup: A copy backup copies all selected files but does not mark each file as having been backed up (in other words, the archive attribute is not cleared). Copying is useful when it is needed to back up files between normal and incremental backups because copying does not affect these other backup operations.
- Daily backup: A daily backup copies all selected files that have been modified the day the daily backup is performed. The backed-up files are not marked as having been backed up (in other words, the archive attribute is not cleared).
- Differential backup: A differential backup copies files created or changed since the last normal or incremental backup. It does not mark files as having been backed up (in other words, the archive attribute is not cleared). When performing a combination of normal and differential backups, restoring files and folders requires that the last normal as well as the last differential backup.
- Incremental backup An incremental backup backs up only those files created or changed since the last normal or incremental backup. It marks files as having been backed up (in other words, the archive attribute is cleared). When combining normal and incremental backups, it is required to have the last normal backup set as well as all incremental backup sets in order to restore your data.
- Normal backup A normal backup copies all selected files and marks each file as having been backed up (in other words, the archive attribute is cleared). With normal backups, only the most recent copy of the backup file or tape is required to restore all of the files.

Windows also backup the non-file data such as; System description of the backup to procure exact replacement in case of disaster, file metadata(permission, ownner, group or ACL) that are required to recreated the original environment while restoring, partition layout. Compression is provided through the storage feature when tape drive hardware is used. Normally in Windows when copied, a file inherit the permission of the destination directory or storage, it

is possible to restricts access to the tape to the owner or members of the Administrators group. File exclusion are when performing backup of en entire directory or partition is possible to configure but only with the backup GUI interface.

3.2.2 Linux Backup:

In Linux, the story is different. Configuration files are text based and(except for when they deal directly with hardware) are largely system independent. In contrast with the Windows backup tool that deals with the details of how the operating system is installed on the system and hardware, Linux backups are about packaging and unpackaging files. In Unix/Linux systems any program that can copy files can be used to perform some sort of backup, there is therefore a bigger variety of commands that can be used in Unix for backup such as *dump, tar, cpio, dd, etc.*.

Linux file systems ext2/ext3 do not have the archive bit available in NTFS but provide three different time stamps (creation, modification, and access) that can be used to work around this. With this time stamps it possible to perform similar backup type such as incremental, daily or weekly performed in Windows with the *tar* command for example. Compression can be done directly with the help of compressing package such as (gzip) as well on the tape drive when supported. File exclusion option is available directly from the *tar* command line.

Chapter 4

Ontology design. Case Study: Backup operation in Windows and Linux

4.1 Tools

4.1.1 *Protégé*

Protégé is a free, open-source platform that provides a suite of tools to construct domain models and knowledge-based applications with ontologies. At its core, *Protégé* implements a set of knowledge-modeling structures and actions that support the creation, visualization, and manipulation of ontologies in various representation formats. It can be customized to provide domain-friendly support for creating knowledge models and entering data. Further, *Protégé* can be extended by way of a plug-in architecture and a java-based application programming Interface (API) for building knowledge-based tools and applications[14].

The *Protégé* platform supports two main ways of modeling ontologies[14]:

- **The *Protégé*-Frames editor:** enables users to build and populate ontologies that are frame-based, in accordance with the Open Knowledge Base Connectivity protocol (OKBC).
- **The *Protégé*-OWL editor:** enables users to build ontologies for the Semantic Web, in particular in the Web Ontology Language (OWL). An OWL ontology may include descriptions of classes, properties and their instances. Given such an ontology, the OWL formal semantics specifies how to derive its logical consequences, that is facts not literally present in the ontology, but deduced by the semantics. These deductions may be based on a single document or multiple distributed documents that have been combined using defined OWL mechanisms. This is the platform that has been use during this work to build ontologies.

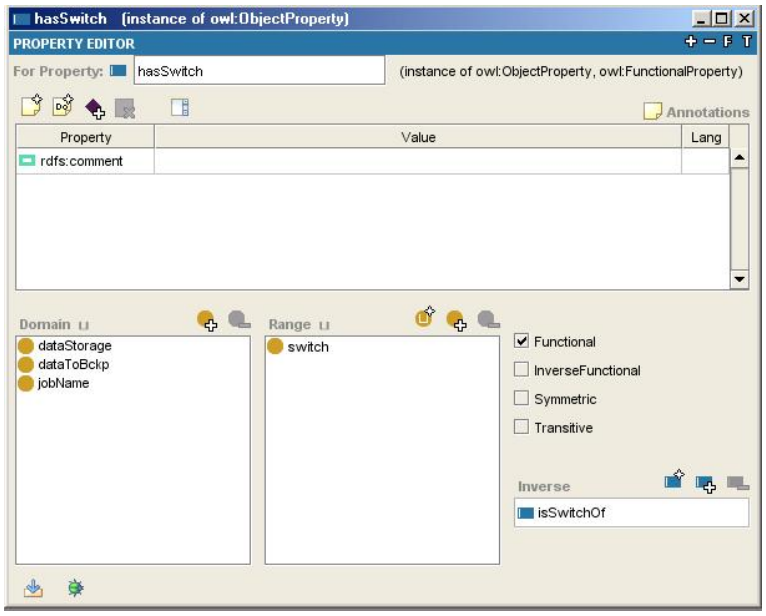

Figure 4.1: Property window in Protg-OWL editor

Protégé-OWL editor was used on this work to build the ontologies. It provides a friendly interface for creating classes, creating properties, associating properties to classes, creating restrictions etc. Figure 4.1 shows a snapshot of the properties creation windows. This interface allow to describe properties characteristics such as *inverse, functional, symmetric, transitive.* Properties link individuals from a domain to individuals from the range, this figure also show that the property ”hasSwitch” links individuals in classes ”dataStorage” ”dataToBckp” and ”jobName” to individuals in class ”switch”. *Protégé*-OWL editor include a facility to generate RDF/XML code for the ontology created with the GUI interface.

4.1.2 The Ontology mapping and merging tool: PROMPT

PROMPT is a suite of tools for multiple-ontology management implemented as an extension to *Protégé* trough a set of plug-ins. It includes the following tools as illustrated also in figure 4.2

- iPROMPT: is an interactive ontology-merging tool. iPrompt leads users through the ontology merging process, suggesting what should be merged, identifying inconsistencies and potential problems and suggesting strategies to resolve them.
- AnchorPROMPT: provides pairs of related terms to help users in ontology mapping and alignment. It provide suggestion to *iPROMPT* to perform merging.
- PROMPTFactor: This tool allow the user to export part on an ontology to another one.

- PROMPTDiff: is a tool for comparing ontology versions

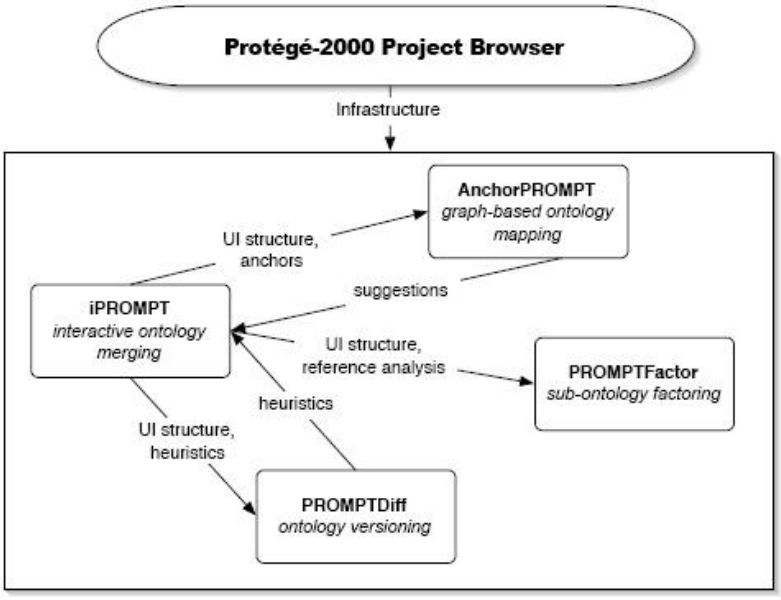

Figure 4.2: The PROMPT infrastructure and interactions between the tools.

AnchorPROMPT was the feature of interest in this work and it's referred as PROMPT in the rest of the this text. *Anchor-PROMPT* produces a set of new pairs of semantically close terms. To do that, Anchor-PROMPT traverses the paths between the anchors in the corresponding ontologies as shown in figure 4.3. A path follows the links between classes defined by the hierarchical relations or by slots and their domains and ranges. Anchor-PROMPT then compares the terms along these paths to find similar terms [15].

4.1.3 Reasoning tool: RACER

RACER (Renamed ABox and Concept Expression Reasoner) commercially known as RacerPro has its origins within the area of description logics. Since description logics provide the foundation of international approaches to standardize ontology languages in the context of the semantic web, RacerPro can also be used as a system for managing semantic web ontologies based on OWL, that is it can be used as a reasoning engine for ontology editors such as *Protégé*. However, RacerPro can also be seen as a semantic web information repository with optimized retrieval engine because it can handle large sets of data descriptions. RacerPro provides the following services for OWL ontologies and RDF data descriptions[31]:

- Check the consistency of an OWL ontology and a set of data descriptions.
- Find implicit subclass relationships induced by the declaration in the ontology.

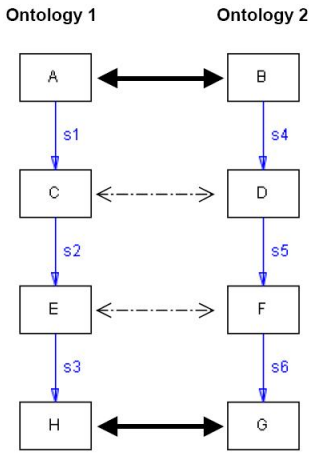

Figure 4.3: Traversing the paths between anchors. The rectangles represent classes and labeled edges represent slots that relate classes to one another. The left part of the figure represents classes and slots from one ontology; the right part represents classes and slots from the other. Solid arrows connect pairs of anchors; dashed arrows connect pairs of related terms[15].

- Find synonyms for resources (either classes or instance names).
- Since extensional information from OWL documents (OWL instances and their interrelationships) needs to be queried for client applications, an OWL-QL(query language) query processing system is available as an open-source project for RacerPro.
- HTTP client for retrieving imported resources from the web. Multiple resources can be imported into one ontology.
- Incremental query answering for information retrieval tasks. In addition, RacerPro supports the adaptive use of computational resource: Answers which require few computational resources are delivered first, and user applications can decide whether computing all answers is worth the effort.

RacerPro has been used in this work mainly for consistency checking of the ontologies, to generate inferred taxonomies and to attempt to induce relationships or restriction between the ontology components. RacerPro is also simply referred as RACER in the remaining of this text.

4.2 Building the ontologies: The methodology

It appears not to be a correct method or an agreed standard for developing ontologies, but rather there are general issues to be considered while developing an ontology. However in [1] and [10] comprehensive methodologies for devel-

oping ontologies have been proposed. In this work we used a combinations of these two proposed methods which are eventually quite similar.

4.2.1 Step One: Purpose and Scope

4.2.1.1 Purpose

What is the backup ontology going to be used for? Section 2.3.2 presented the different usage or role of ontologies. The purpose the ontologies created is to describe the backup syntax of Windows (using the *ntbackup* utility) and Linux(using *tar* program) in order to be able to create mapping between both representations. The ultimate goal is to be able to perform backup through high level requirements using a general purpose backup Ontology without the details of underlying syntaxes. The Mapping between the two ontologies contribute to the translation process of requirements between the two system to perform the backup operation. These representation could be used to exchange information between different system management tools.
One of the ways to determine the scope of the ontology is to write down a list of questions that a knowledge base based on the ontology should be able to answer[10]. This is referred as *competency questions*.What are the question the backup ontology should answer? Or what type of high level requirement should the ontology find answer for?

- We need this operation, find the appropriate syntax and do it.
- what is the meaning of this information received by this SM tool.
- I want to do a backup every day at mid day and exclude the file x from it.
- I want to do incremental backup of data x every one hour
- I want to perform a backup of file x every more additional 2mb to the file.
- I want to make a backup of this file exactly at time x every 2 days but not in the weekend.
- etc....

There is also a lot of discussion about types of ontologies as it has been discussed in section 2.3.3. According to the classification of ontology types described by Mizoguchi in [16], the ontologies build on this work can be referred as task dependent domain ontologies as the representation is only related to the backup task.

4.2.1.2 Scope

The scope is about characterizing the range of the intended users of the ontology. It provides the answer to the question: who/what is going to Use the

ontology? The intended users of the ontologies created are machines, but representing these ontologies with visualizing tools it can be equally useful to system administrators to understand syntax requirements for backup operation in both operating systems.

4.2.2 Step Two: Considering re-using an existing Ontology

There exist already many libraries of reusable[1] ontologies available in electronic format that can be imported to en ontology development application. This can be a requirement if some of the a system requires to interact with another application that has committed to different ontology. For the case study of this work, no similar ontology was found. However to include the "time" concept in the created ontology the already existing "Time Ontology in OWL"[2].

Ontology Capture:
After the two above steps, the ontology capture was done. Ontology capture involves the following steps:

- Enumeration of terms or concepts in the domain(Backup operation and syntax)
- Class definition and class hierarchy definition
- Classe's properties definition
- Instances creation

4.2.3 Step Three: Enumeration of important terms or concepts

This steps is about writing down all the terms in relation to performing backups for both Windows and Linux (using the *tar* command) operating systems. Command help page for Windows and man pages(for *tar*) have been used to collect the information. The name of term are given as close as possible to their meaning to avoid confusion or unnecessary explanation of terms. The aim of this step is to provide a comprehensive list of terms without taking care of overlap between concepts they describe, relations among the terms or the properties that the concepts could have. The terms represented were later categorized as classes, properties or instances.

The following terms can be enumerated for a general purpose backup operation:

[1]DARPA agent markup language (DAML) ontology library is available at http://www.daml.org/ontologies/
[2]Ontology of temporal concept at http://www.w3.org/TR/owl-time/

backup command	file to backup	tape	storage file	incremental
compression	files to backup	storage file	update	differential
system backup	directory	storage path	normal backup	permission
backup time	hard disk	file	file path	backup description
SAN storage	tape name	tape	size of data	backup frequency

Below is a list of terms for backup under Windows system.

ntbackup	file to backup	tape	file to backup	incremental
backup	file backup list	storage file	update	differential
systemstate backup	directory to backup	storage path	copy	permission
time and date	directory	file	network path	normal
compression	media description	pool name	media name	daily

A list of terms for Linux backup with command "tar" is shown on the table below

tar	archive name	archive path	file to backup	file to backup list
file to backup path	append backup	create archive	update	file exclusion
compression	data verification	incremental	file age	permission
time and date	directory	file	size	access-time

4.2.4 Step Four: Defining Class and Class hierarchy

Building the class hierarchy is about organizing concepts following the "is-a" relation: a class A is a subclass of B if every instance of A is also an instance of B. From the list of terms enumerated in the previous section we selected the terms that have independent existence and could represent objects rather than terms that describe objects. These terms are classes in the ontology and are making up the class hierarchy. There are three main approaches in developing a class hierarchy[1]:

- The top-down approach: Which starts by first defining the most general concepts in the domain of interest and subsequently the the most specific concepts.
- The bottom-up approach: in this approach the development process starts with the definition of the most specific classes, and leaves of the hierarchy, with the subsequent grouping of these classes into a more general concepts
- A combination of both top-down and bottom-up approaches.

Applied to the case study, the top-down approach was used. The command line syntax information were categorized using the simple categorization found in help files that is: "commands", "data to backup", "option(or parameters)" and "data storage". The same approach was used for both Windows and Linux.

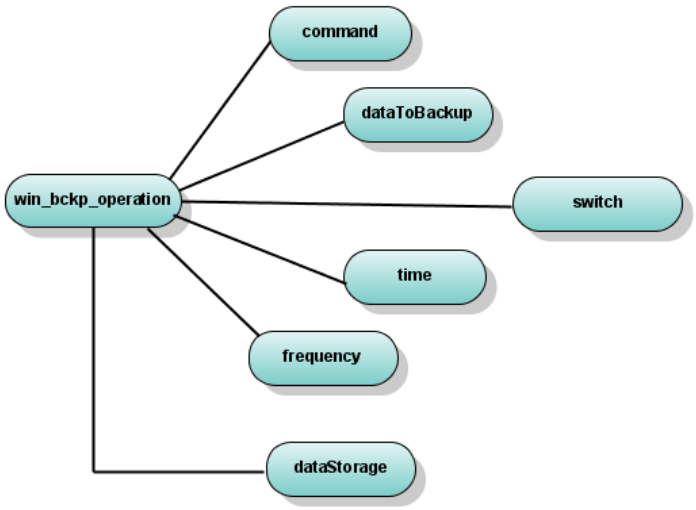

Figure 4.4: Windows backup top level classes

4.2.4.1 Windows backup class hierarchy

In Windows we can group the backup operation into four high level classes:

- command class: that includes two instances *ntbackup, backup*
- dataToBackup class: Which includes include fileName, fileListName and directoryPath sub-classes
- dataStorage class: It includes diskStorage and TapeStorage sub-classes.
- switch[3] class: consists of instances representing the different command line switches (/p, /s etc.)

Two optional concepts (time and frequency) were added to the hierarchy as shown in figure 4.4. In Windows this concept are not specifically connected to the command line syntax but can be added with the scheduling command *at*. The *time* class represent the moment at which a backup operation can be started or stop for example. The *frequency* represents how often an operation is repeated, that is every x minute(s), x hour(s), x week(s) etc. The time class appear to be actually more relevant to the backup under Linux due to the presence of more options using the time and date information attached to files. It most be specified that the "winBackupOperation" class shown in figure winclass does not represent the superclass of the others classes. It is used to represent the domain of interest. More hierarchies of this top level classes can be seen on the figures **?? ??**. .

[3]switch refers here to command line option swicth such as /A /F etc.

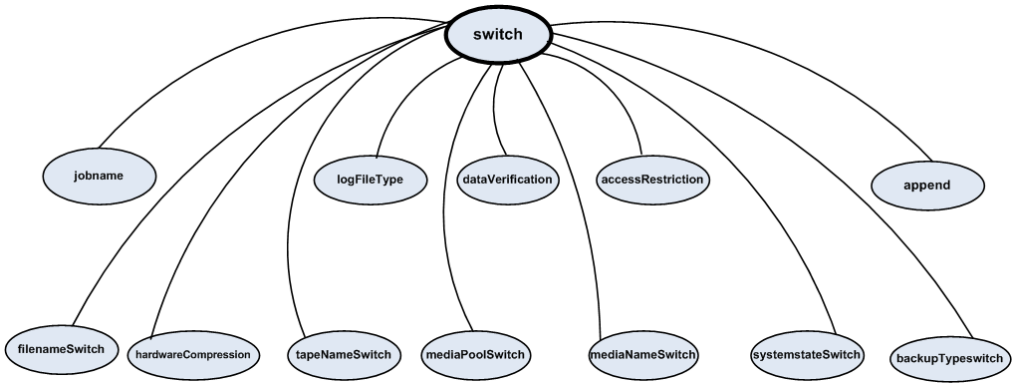

Figure 4.5: Windows backup command line taxonomy for parameters

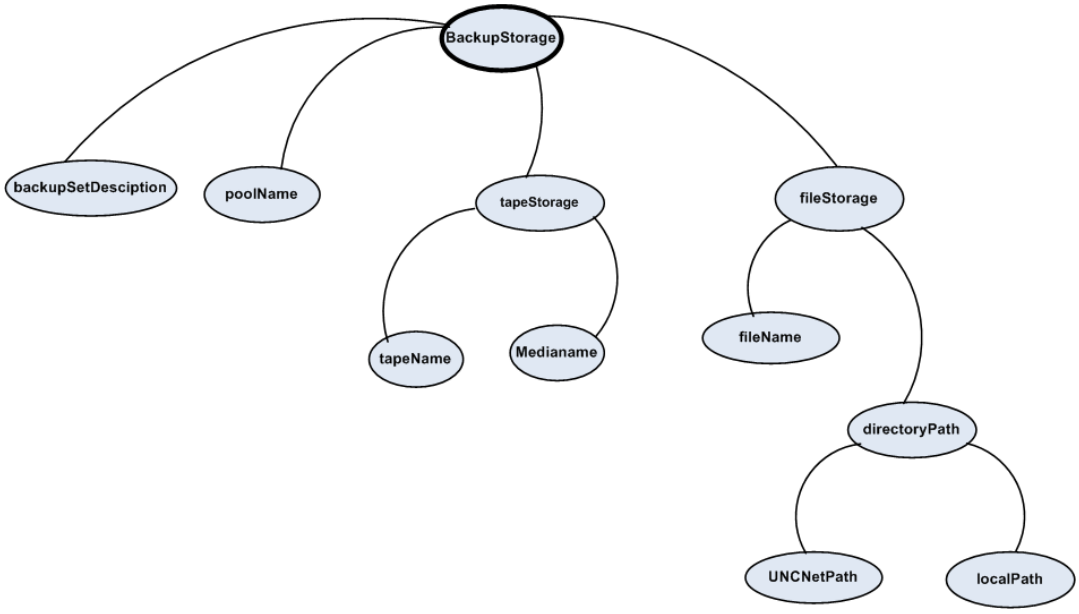

Figure 4.6: Windows backup command line taxonomy for backup storage

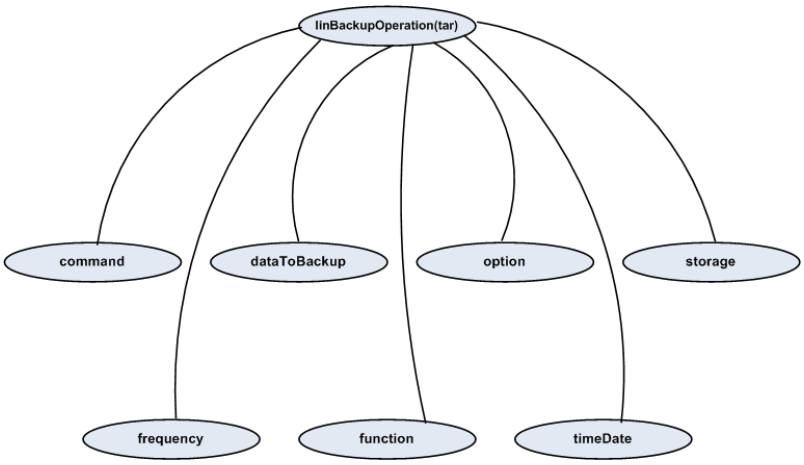

Figure 4.7: Linux(tar) backup top-level classes hierarchy

4.2.4.2 Linux backup class hierarchy

The classes in for the backup with *tar* program were described as follow(figure 4.7:

- Command: This class will include a single instance that is the command *tar*
- storage: which consist of the ”archiveName class” (the tar file name under which data are backed up), and ”archivePath class”(directory in side which data is backed up) and the ”tapeFileName” (tape device name for backup)
- dataToBackup: consists of ”fileName”, ”fileNameList” and ”directoryPath” classes.
- function: represent the class of the necessary options. At least one instance of this class should be included in the command line.
- option: This class consists of instances representing optional options to the command line.

This hierarchy is similar to the Windows hierarchy described earlier with slight terminological divergences which do not however present the implementation differences. Classification of some of the other classes are shown in the figure 4.8

4.2.5 Step Five: Defining properties of Classes

Now that the classes have been created, the internal structure of these concepts are to be specified through properties. Properties are also known as *slots* in *Protégé* and are equivalent to the UML's *relations*. Properties describe classes and provide relationship between instances of a class with others. These properties provide information that are needed to fully represent the knowledge

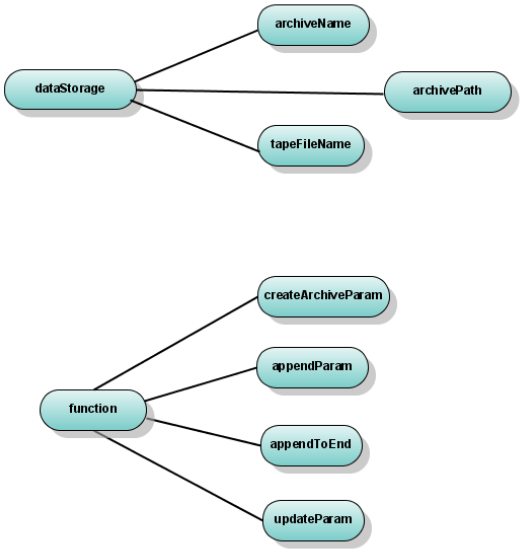

Figure 4.8: Linux(tar) function and dataTobackup classes taxonomy

in a domain. Some of the terms listed in section 4.2.3 could represent properties of some classes. As example the term ”size” could represent a property of a ”fileName” class. On another scenario ”fileName” itself could have represented a property of a class ”file”. On the scenario of this work the file name through the ”fileName” class represent the object of interest itself for the backup operation and therefore it was considered as a class. While defining properties with *Protégé* several information needed to be to specified such as:

- Type of the property:
  Protégé provides object, datatype and annotation properties. Object properties are used to link an instance to another instance while datatype properties link an instance to an XML Schema Datatype value. Annotation properties were not used during this work. The property shown in Figure 4.9 is an example of object type property that were defined for this work.

- Characteristics of the property:
  Characteristics augment the meaning of the properties. There are, *functional, inverse, symmetric and transitive* characteristics. A functional property relate on instance from a particular class with only one instance in another class. This means that in case a functional property relates an instance with two different instances then the later will be consider as equal by the inference mechanism of *Protégé*. Two properties are inverse if there are reciprocal that is they are applied between the same instances but with opposite meaning and direction as shown in figure 4.10. A property is said to be symmetric if it can be applicable to the two instances in both direction with the same meaning as shown in figure 4.9.

Transitive properties have not been used in this work.

- Domain and Range classes of the property.
  Properties link instances in a *domain* to instances in a *range*. That is if we consider a relationship such as:

  (boot.ini) "***hasPath***" (C:)

  The class "dataToBackup" to which belongs the file boot.ini represents the *domain class* while the class "directoryPath" to which belongs the instance path "C:" is the *range class* for this property

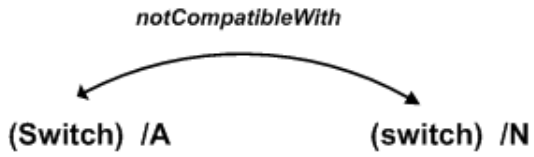

Figure 4.9: *notCompatibleWith* is a symmetric property between the two instances

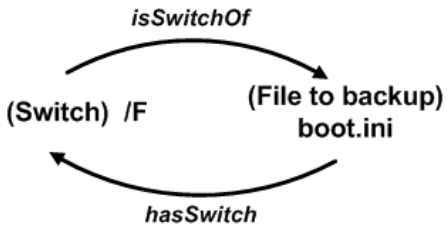

Figure 4.10: *isSwitchOf* and *hasSwitch* are inverse properties

Table 4.1 shows some of the properties that have been created to describe the relationship in the Windows backup ontology

property Name	**Domain**	**Range**
hasPath	fileName	directoryPath
isPathOf	directoryPath	fileName
isSwitchOf	Switch	dataStorage, dataToBackup
hasSwitch	dataStorage,dataToBackup	Switch
compatibleWith	switch	switch
notCompatibleWith	switch	switch
precede	switch	dataStorage
follow	datastorage	switch

Table 4.1: Properties list for the windows Backup Ontology

4.2.6 Step Six: Defining and describing classes with properties restrictions

In OWL properties are used to create *restrictions*. Restrictions are used to restrict the individuals that belong to a class. *Protégé* provides an good facility for creating restrictions through its interface. There are 3 categories of restriction that can be defined with *Protégé* [17]:

- Quantifier restriction: Which includes the existential quantifier(∃) and the universal quantifier(∀)
- Cardinality restriction: Cardinality restrictions are used to talk about the number of relationships that an individual may participate in for a given property.
- hasValue restriction: Describes an anonymous class of individuals that are related to another specific individual along a specified property.

4.2.6.1 Quantifier restrictions:

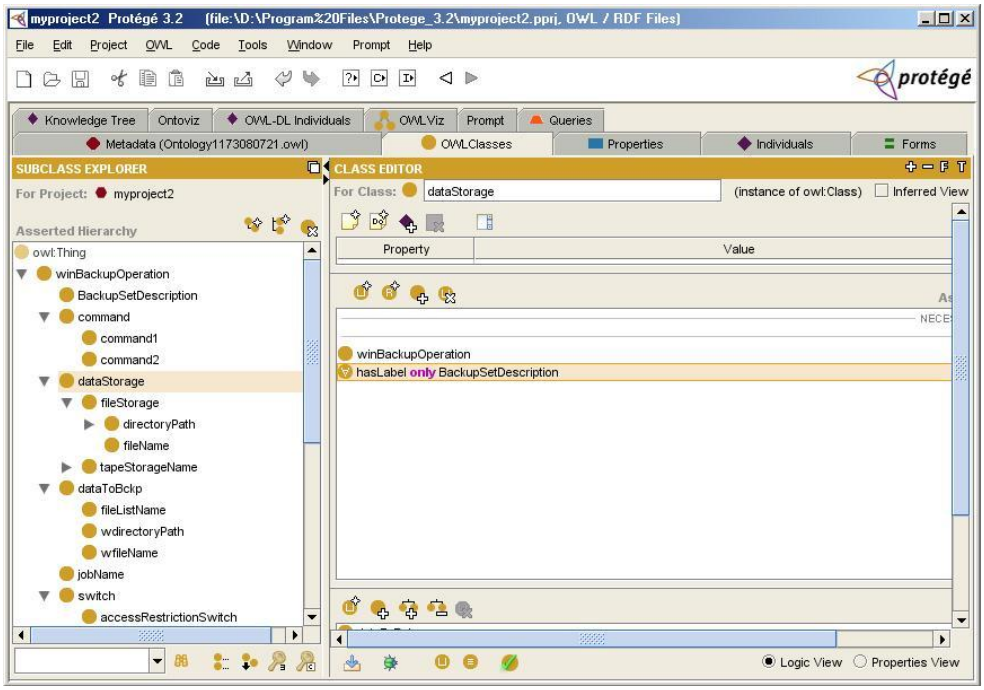

Figure 4.11: creating restriction with *Protégé*: here the universal restriction is applied to the class "dataStorage"

These types of restrictions are composed of a quantifier, a property, and a filler. There are two types of quantifier:

- (∃) the existential quantifier:
  Which can be read as *at least one* . It describe the set of individuals

that have at least one specific kind of relationship to individuals that are members of a specific class. Below is an example of restrictions:

$\exists$ *isSwitchValue* "accessRestrictionSwitch"

Applied to the class "switchValue", means that instances in the class "switchValue" class have at least one relation with an instance in the "accessRestrictionSwitch" class or with any other class. This is to reflect the fact that in the Windows backup command line the command line switches /V, /R, /RS require a "yes" or "no" to enable or disable the option. The idea behind this relation is to represent the following:

"yes — no" isSwitchValue /V $\Rightarrow$ /V:yes or /V:no.

- ($\forall$) the universal quantifier:
  Which is read as *only*. Universal restrictions portray the set of individuals that, for a given property, only have relationships to other individuals that are members of a specific class. Figure 4.11 depicts the asserted condition window in *Protégé* from where restrictions are applied to classes. As shown in figure 4.12, 0 or more instance(s) of the class "dataStorage" have the relationship *hasLabel* **only** with instances of the class "BackupSetDescription".

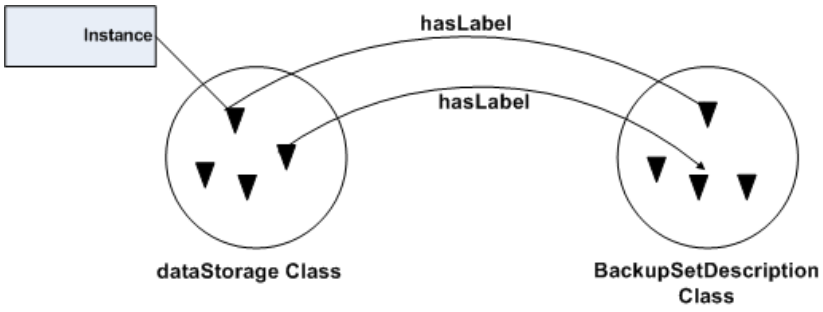

Figure 4.12: Diagram describing the Universal restriction ($\forall$)

4.2.6.2 Cardinality Restriction.

Cardinality restrictions are used to talk about the number of relationships that an individual may participate in for a given property. Cardinality restrictions are conceptually easier to understand than quantifier restrictions, and come in three types:

- Minimum cardinality restrictions($\leq$):
  which specify the minimum number of relationships an instance must participate in for a specific property. As an example a we can have the

same file name in different directory, which means a file name should have a least one directory path. As a ”fileName” class was defined with a relationship *hasPath* with ”directoryPath” class, a minimum cardinality restriction was defined for instances in the ”fileName” class with respect to the *hasPath* property. The code below shows how this restriction is expressed in RDF/XML format generated from *Protégé*

Example 1 — *XML/RDF generated code showing expressing information define from Protégé GUI*

```
<owl:Class rdf:ID="fileName">
       <rdfs:subClassOf rdf:resource="#fileStorage"/>
       <rdfs:subClassOf>
           <owl:Restriction>
               <owl:onProperty rdf:resource="#hasPath"/>
               <owl:minCardinality rdf:datatype="&xsd;int">1
</owl:minCardinality>
           </owl:Restriction>
       </rdfs:subClassOf>
       <owl:disjointWith rdf:resource="#directoryPath"/>
   </owl:Class>
```

- Maximum cardinality restrictions ($\geq$):
  which specify the maximum number of relationships an instance can participate in for a specific property.
- Cardinality restrictions(=):
  specify the exact number of relationships and instance can participate in for a given property

4.2.6.3 hasValue Restriction ($\ni$)

This restriction describe a class of instances that are in relation with a specific instance which might belong to any class. As an example to specified the command option switch to be used with a storage type (file or tape) the following restriction were applied respectively to the classes ”fileStorage” and ”tapeName” and ”mediaName” (under Windowns command line the option switch /J , /T and /N are respectively used before specifying a file name, tape name or media[4] name as backup storage). A schematic representation of the meaning of these restrictions is shown in figure 4.13

$\ni$ hasSwitch *J*
$\ni$ hasSwitch *T*
$\ni$ hasSwitch *N*

[4]In Windows terminology it refers to an empty tape disk

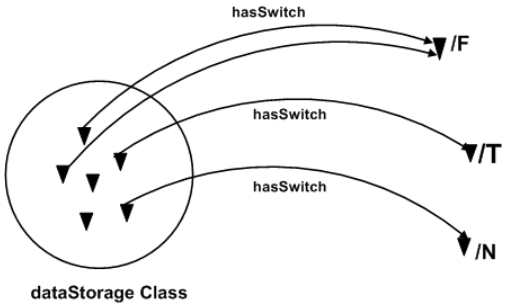

Figure 4.13: Meaning of the *hasValue* restriction(∋): Instances(specific files, tapes) from the ”fileStorage” class are used with specific option switches(/F, /T, /N) in the ”switch” class

4.2.7 Mapping the Ontologies

As the different ontologies have been created. The next stage was to performed a mapping using the PROMPT plugin of *Protégé*. There were two main objectives at this stage. First to appreciate how efficiently the PROMPT performed the automatic mapping between the two ontologies and second and not less important to understand what kind of mapping is done. That is investigating the reasoning used for this mapping. As mentioned earlier ontology mapping for interoperability is often performed through a common ontology which is mapped with the different ontologies individually rather than mapping each ontology to the other directly. On this work the creation of the common ontology was overlooked and only the mapping between the backups ontologies for Windows and Linux was performed.

PROMPT mapping tool offered different algorithms to perform the mapping:

- Lexical mapping: Which suggests mapping based on lexical similarities.
- Framework for Ontology Alignment and Mapping[5] (FOAM):
  This algorithm is based on heuristics (similarity) of the individual entities (concepts, relations, and instances). As result it provides pairs of aligned entities. The FOAM plugin for PROMPT could not be used in this work as it was failing with unexpected errors when started due to a bug specified on *Protégé* website[6]. However the fix proposed could not be fix successfully during this work.
- Using UMLS:[7] This algorithm could not be used.

Therefore the lexical mapping algorithm was the only algorithm that has been used to perform the mapping with PROMPT. Figure 4.14 shows the automatically ”suggested” mapping between the two ontologies. As it can be seen in the figure the list of suggested mapping is relatively small: The remaining

[5]http://www.aifb.uni-karlsruhe.de/WBS/meh/foam/
[6]http://protege.cim3.net/cgi-bin/wiki.pl?Prompt
[7]Unified Medical Language System concept identifier for matching

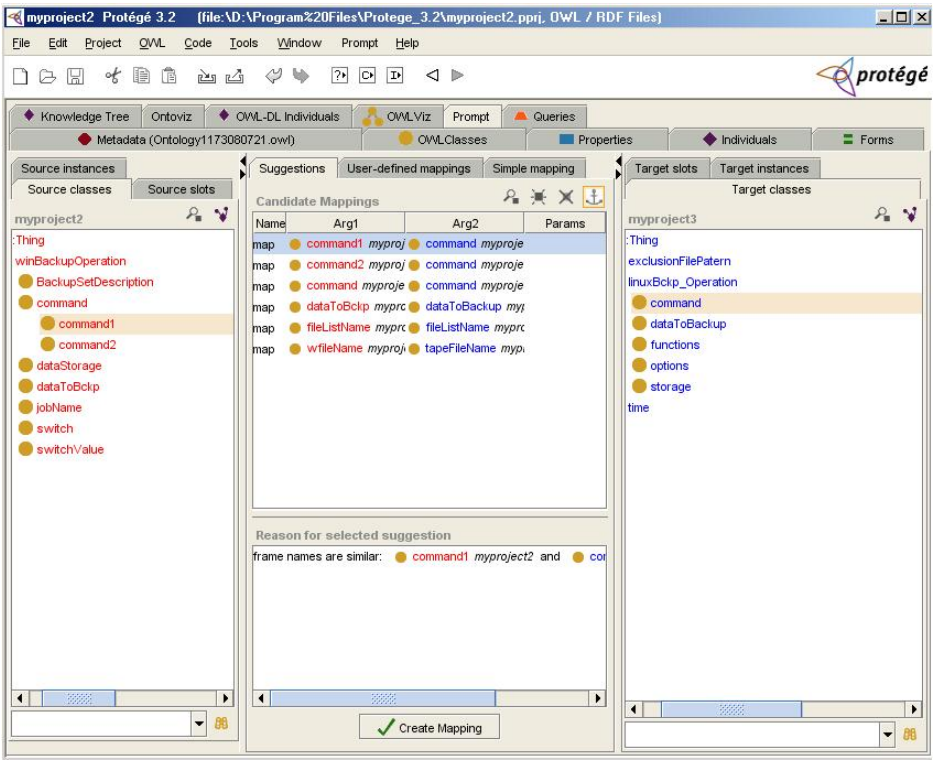

Figure 4.14: mapping suggested automatically by PROMPT

mapping were assessed and mapped manually. Classes , properties and instances were mapped in a one-to-one fashion as well as one-to-many. Below are a list of mapping.

Chapter 5

Result Evaluations and Discussion

This chapter evaluates the ontologies created as well as the mapping results obtained with PROMPT. It also includes discussion about the worthiness of using the ontology approach to solve the interoperability problem presented in the backup study case in particular and in system administration in general.

5.1 Ontologies evaluation

It seems there is no standard yet defined for evaluating and validating ontologies although several methodologies have been proposed. [18] suggests an interesting and complex methodology for performing this task. The evaluation was done mainly by a combinations of these suggestions and investigating the correctness of the knowledge represented by the ontologies by trying to find answer questions such as:

- Are there hidden assumptions?
- Could a non-expert capture the knowledge represented?
- How efficiently could the representation being used by automated agent (such configuration tools) ?
- Are there easily modifiable or re-usable?

5.1.1 Hidden assumptions:

An ontology is an ”explicit” specification. Although most ontology development tools include reasoning functionality to deduce knowledge it important to provide as much as possible relevant information to describe the specific domain. In both the Windows and Linux(tar) command line backup syntax, the different parts of the entire syntax were represented and their relationship. The task appeared to be an ontology capture of *tar manpage* for Linux or the

ntbackup help file. While it seemed straight-forward to describe parts of the command line sections (command itself, data to backup, storage location of backed up data, command line options), the order of usage of these sections had to be specified as well. Details of representing parameter switches format or their incompatibilities had to specified. In the example of Windows backup parameters present different formats:

```
/switch <file name with path>
/switch <name>
/switch:{yes | no | on | off|f|s|n}
```

The ordering was made using properties such as *precede* or *follow* to say for example that the switch /F *precede* the name of the destination file of the backup operation. The tradeoff of an exhaustive insertion of properties and relationships is the risk of building an heavy ontology which makes little use of reasoning capabilities, increasing as well the risk of inconsistency. Fortunately *Protégé* provides a consistency checker that was executed upon the ontologies. Optimizing the amount of information included was not done during this work. The idea behind this optimization is about an efficient usage of the reasoner ***RACER*** in order to remove relationship that could have been deduce through properties characteristics such as symmetry, functional, inverse or transitivity.

5.1.2 Quality of the knowledge capture and Usability

Ontology is *a shared understanding* and at the same time ontology design is a subjective activity which means that the same ontologies created in this work would be done differently by different people. However ontology engineering as already mentioned includes mechanism to map or merge different ontology even from the same domain.

Can these ontologies being understood by non-experts of the domain through a graphical representation? We aimed at reducing terminology differences between Windows and Linux but the ontologies includes already these differences: ”switch” class for Windows as compared to ”option” class for Linux which both represent command line parameters classes. It seems therefore that the ontologies interpretation from a human perspective might be ambiguous which is in contrast with the goal of using ontologies. Maybe the question is instead not appropriate for the scenario of this work, as ontology concept has mostly evolved from the field of Artificial Intelligence with the aim of achieving share knowledge and enabling reasoning between computer agents. As suggested in [19] ontololgy could be used for standardization of terminology by creating a common vocabulary for communication. This could have been achieved better by finalizing a common ontology resulting of the merging of the two ontologies as initially planned. Therefore a more important question is to know how could these ontologies being used by automated agents (system management tools).

The implementation of usage of these ontologies by a software was not done in this work. Nevertheless a software tool could extract the information from the XML/RDF code generated by *Protégé*. Additionally *Protégé* includes plug-ins for several querying language such as SPARQL[1]. The ontologies could be exported to a relational database in which each class would represent a table hence making the use of standard SQL queries although this type of approach might results in lost of semantic in data representation as suggested in [20].

5.1.3 Querying the ontologies

Protégé provides plugin for SPARQL as well as a graphical user mode query tab. The latter appeared simpler to use although it could only query instances from the ontology. Some relationships were tested such as the *NotCompatible-With* property between options switches as shown in figure 5.1. Unfortunately this the query tab was not flexible enough to build interesting queries while SPARQL on the other hand was rather difficult to use. With SPARQL however queries were performed over classes and properties. below a simple query to enumerate the list of subclasses of the ”switch”class in the windows ontology.

```
SELECT ?x
WHERE { ?x rdfs:subClassOf:switch }
```

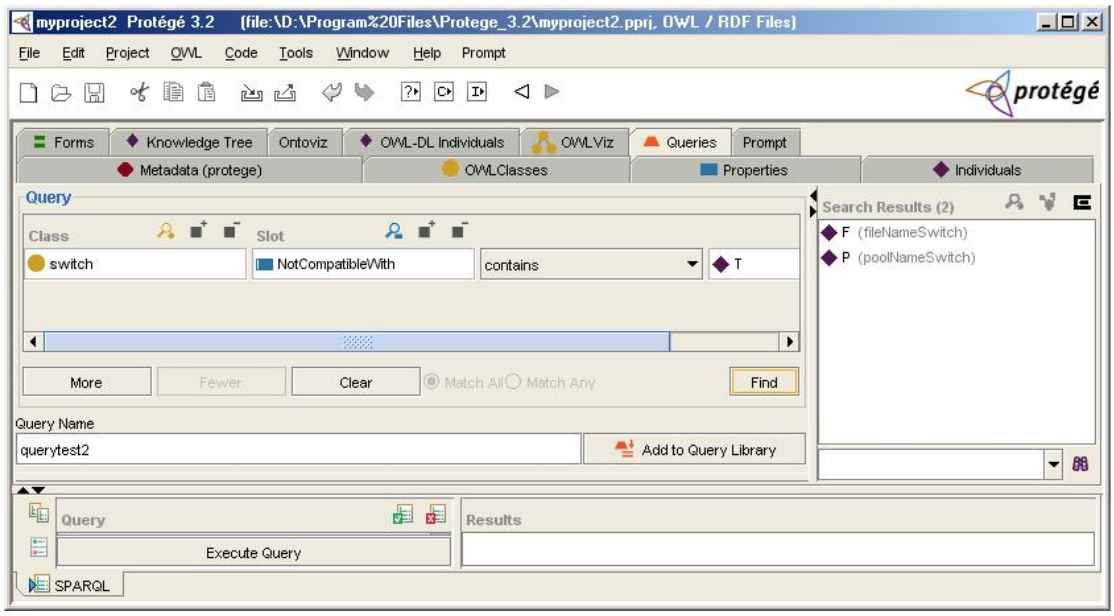

Figure 5.1: Query result of parameters switches non compatible with the parameter switch /F in Windows

[1]SPARQL is an RDF query language; its name is a recursive acronym that stands for SPARQL Protocol and RDF Query Language.

The evaluation provided above is rather intuitive than systematic. In [18] Aldo Gangemi et al proposed a framework for evaluating and validating ontology.

5.1.4 Suggested framework for ontology evaluation

[18] proposed measurement method for ontology. It evaluates ontology by introducing specific measurement types and metrics.

- Structural evaluation:
  This about measuring the topological and logical properties of an ontology when represented as a graph. They defined metrics such as *depth, breadth, leaf and sibling distribution, density, modularity, consistency, complexity, logical elements distribution etc.*
- functional evaluation:
  Related to the intended use of a given ontology and of its components through accuracy investigation from domain experts.
- Usability evaluation : depends on the level of annotation of a given ontology. How easy it is for users to recognize its properties?

The mathematical model of the different metrics used in this proposed framework have not been used in our work due to time constraints.

5.2 Mapping evaluation

5.2.1 Level of automation and accuracy

The PROMPT algorithm used during the mapping was the Lexical based algorithm. The mapping process provided by *Protégé* is semi-automatic as it provides suggestion and requires user intervention to complete the mapping as well as defining the remaining possible mapping manually.As shown in figure 5.2 the mapping suggested by PROMPT was not always accurate.

Two metrics proposed in [21] which are standard information retrieval metrics have been used to evaluate the accuracy of mapping of the two ontologies:

1. Recall: describes the number of correct mappings found in comparison to the total number of existing mappings. The total number of possible mapping include only specifically defined classes and properties (excluding inferred datatypes or built-in properties)

$$recall = r = \frac{Correct\_mapping\_suggested}{all\_possible\_existing\_mappings^2} \Longrightarrow r = \frac{6}{17} = 0.35 = 35\% \tag{5.1}$$

2. Precision: measures the number of correct mappings found with respect to the total number of suggested mappings (both correct or wrong).

$$precision = p = \frac{correct_mapping_suggested}{all_suggested_mappings} \Longrightarrow p = \frac{6}{11} = 0.54 = 54\% \tag{5.2}$$

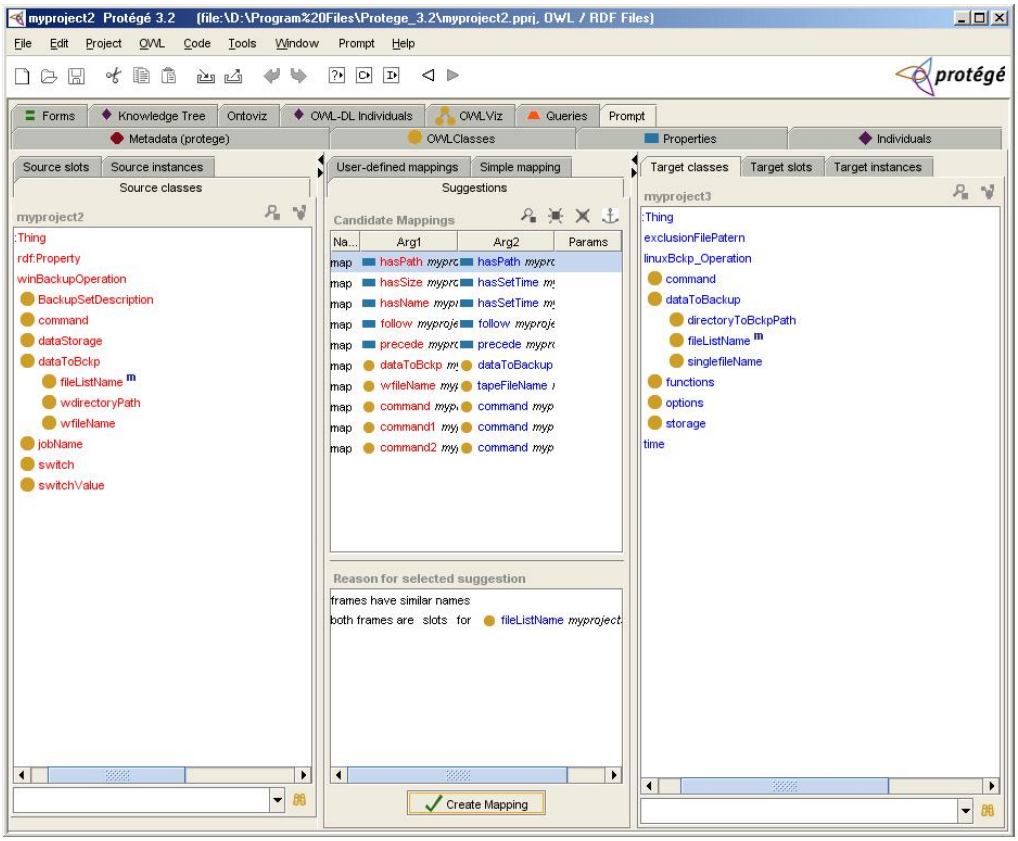

Figure 5.2: mapping suggested by PROMPT including some wrong suggestions such as mapping properties *hasName* and *hasSetTime*

5.2.2 Validation and Problems

The mapping was done on classes, properties and instances. The instances mapping was mainly about matching Windows parameters switches to Linux(tar) options. The list in table 5.1 shows a list of instances mapping that was done. Although the mapping was mainly done manually, *Protégé* allows the mapping to be saved for future retrieval. This is important as retrieval of mapping information could help in the process of translating commands from both systems. *Protégé* offers two types of mapping storage ontology; *Simple mapping* ontology and *Domain_PSM*[3] ontology. The latter ontology mapping was used as it defines the type of instance-level and slot-level mapping relations that

[3]Problem-Solving Methods

you can create between the classes and slots of the domain ontology and the classes and slots as shown in figure 5.3. Figure 5.4 shows the result of querying the Domain_PSM mapping ontology for a similar instance to the Windows verification option /V (represented by in the "dataVerificationSwitch" single instance class) in the Linux(tar) ontology resulting to the *tar* verification option -W (represented by "verificationParam" single instance class).

Table 5.1: Some mappings

***ntbackup*(Windows)**	***tar*(Linux)**	**Remarks**
/*V*	-W	verification option
/*F*	-f	file name option
/*A*	-A	appends to storage option
ntbackup	tar	backup command
/*R*	-p	restriction on storage file
no equivalent	-X	file exclusion option
/*N*	no equivalent	Tape media name
/M{incremental}	no straight equivalent	implementation difference

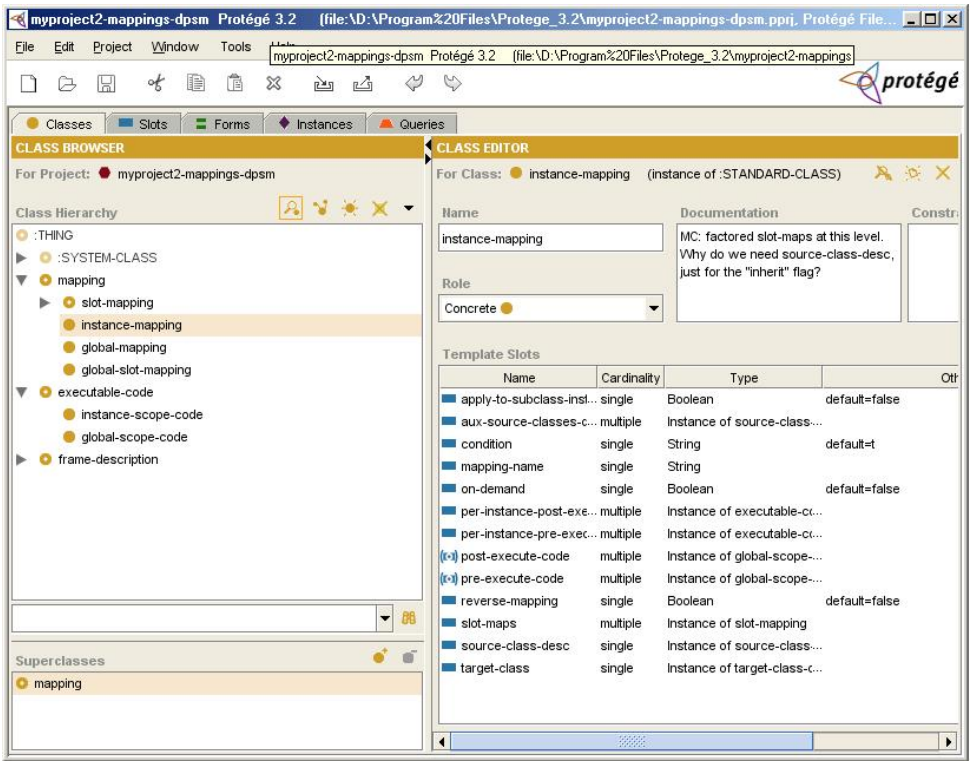

Figure 5.3: psm mapping ontology class browser

There have been however several problems regarding this mapping:

1. Missing concepts:

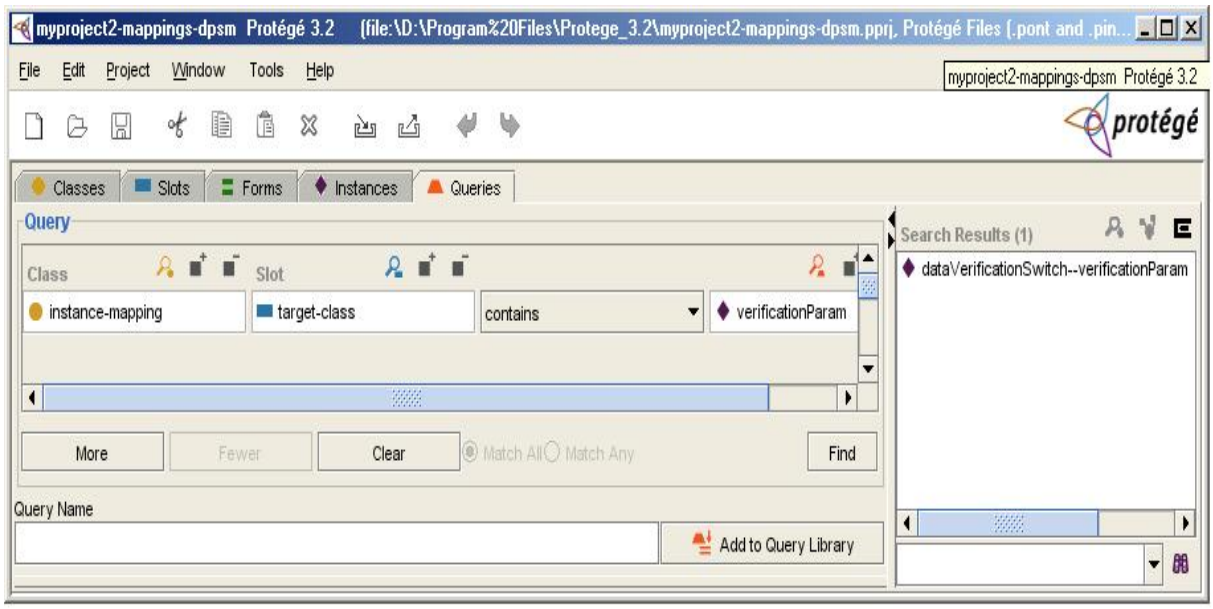

Figure 5.4: Querying psm mapping ontology: Verification option(/V to -W)

There are several concepts that are specific of one system and do not have similar objects in the other system. It was not possible to map options such as; *tar*(Linux) option for file exclusion (-X) or media name parameter switch (/N) for Windows. This problem could have been minimized by creating a common ontology resulting from merging the two ontologies and then performing mapping to both ontologies from this common ontology as originally planned(figure 1.1). As mention earlier this steps has not being undertaken in this work due to time constraints.

2. Difficulties in representing one to many or many to one mappings:
these type of mappings raised some concerns:

   (Windows File name option switch) /F $\Longrightarrow$ -f (tar)
   (Windows tape name option switch) /T $\Longrightarrow$ -f (tar)
   (tar) -f $\Longrightarrow$?

   This is caused by the fact that Linux mounts(represent) devices such as tape disk to file and therefore the *tar* option -f is also applicable to tape disk as well as files on the physical hard drive while in Windows backup uses different option switch for files and tapes.

3. Multi-function commands: Commands in Linux usually have different functions and are used for different purposes. *tar* command by itself can be used for performing backup, compressing and decompressing files. Therefore mapping the *tar* command to the *ntbackup* command makes only sense in the current scenario. This infer than a mapping is related to a context, hence making a ”general” ontology for system administration

a complex task, but instead task oriented ontologies more appropriate.

5.3 Discussion

Earlier in this text we have presented the possible benefits of the ontology-based approach in representing and integrating different knowledge representation. With respect to the study case presented, the following advantages have been considered:

- **Clear representation of concept:**
  Despite a minimum knowledge of the ontology development tool *Protégé*, the features used to build the ontologies were solid enough to clearly represent the backup command line requirement for both systems. Facilities such as the consistency checker have helped to ensure of a consistent representation and the inference facility with the help of the inference engine *RACER* provided the reasoning for deducing properties or restrictions. Inferred taxonomies by RACER for both ontologies are shown in the appendix.

- **Mapping and Merging:**
  These processes are central to integration or interoperability. The PROMPT plugin in *Protégé* presented a good potential in performing mapping although most of the different algorithm could not be tested in this work.

- **Code generators**: RDF/XML code of the ontology were easily generated from the GUI interface of *Protégé* (see Appendix). This code can be fed as input to an program to extract the information or generate classes for application in other programming language such as Java. Several other code generators available in *Protégé* have not been used as they were not relevant to the work (java Schema Classes, *Protégé*-OWL java code etc.).

However there have been several difficulties in using this approach to solve the problem investigated in the study case:

- **Complex querying language:**
  The query tab provided in *Protégé* lacks flexibility for defining user queries and the other query languages such as SPARQL appeared complex to use. The ontologies could have been exported to a relational database to perform queries with standard SQL but this option has not been undertaken due to time constraints. A thorough querying of the ontologies would have been good to better validate and evaluate the quality of the ontologies, rather than the few queries that have been made.

- **Limited Mapping:**
  The complexity understanding the tool *Protégé* and getting the best out of it features have taken a considerable time. There are several features that

should have been overlooked to focus on the mapping issue which represent already by itself a reasonable amount of work. Because of this the use of several mapping algorithms with the aim of getting the best mapping suggestion could not be done. The work relied on lexical mapping algorithm in PROMPT which means that the *recall* and *precision* metrics computed in the mapping evaluation section could have led to worse results if the ontologies were created by two different persons instead of the author.

- **Limited solution:**
  This work is not providing a solution but could be inserted has a components of a complete solution. Beyond merely mapping different commands and options they must exist a back-end application that will implement functions in order to interact with the ontology to retrieved information for the required task on the different systems. This work focuses in only a single command in Linux for performing backup, in a more generalized scenario it would have been required to map the Windows backup concept to several additional Linux based backup programs such as *dump,dd,cpio etc.*. In this type of scenario where a high level requirement(say backup requirement for example) is mapped to different set of commands there would be a need for a ”policy based management system” to evaluate the impact of each set of commands in the corresponding system before implementing it as suggested in [13]

Chapter 6

Conclusion and further work

This work investigated the possibility of an ontology based approach to achieve cross platform system administration in heterogeneous environment. The work has been confined to the backup case study in Linux and Windows environment. The aim was to study how ontology concept would provide both syntactic and semantic mapping between the command lines requirements of both systems. The ontology representations of the command line requirements for the backup task in both Windows and Linux systems, are merely an illustration of the command syntax components and their relationships. These representations can be considered as machine readable manual pages (man page) for the corresponding commands. The mapping performed between the two ontologies created, highlighted the difficulties of accurate and automatic mapping of different ontology representations. This mapping suggested correspondences between both representation regardless of the implementation differences between both systems. This was the case of the difference of implementation of incremental backup in both systems. Certainly the ability to identify implementation similarities or differences is important for automatic mapping which could be a requirement for automated agents relying on ontologies representations to achieve interoperability. A condition for this to happen is to provide the ontology with the semantic describing these implementations. However this work has emphasized on providing semantic of the command line syntax of the backup programs in both Windows and Unix rather than the implementation details. Nevertheless the objective of cross platform administration is to be able to manage simultaneously different systems from a single management interface and translating commands details between diverse systems is consequently a step towards this integration goal.

Although the use of ontology to perform commands mapping might appear to be an excess task as one could suggest alternative such as manual mapping using relational database, ontology representation is clearly a better framework for coherent knowledge representation and sharing with reasoning capabilities which are not available in relational database. There have not been many work related to using ontology concepts in system administration field as it has been done in other field such as semantic web, telecommunication or

autonomic networking. Because of this little interest in ontology for system administration this work could inspire more substantive work such as implementing a complete system including a management interface which interact with the ontology to allow commands to be translated and executed in the target hosts. Similar studies could also be extended to other common system administrator tasks such as files permissions or users management.

Bibliography

[1] Mike Uschold, Michael Gruninger. *Ontologies: Principles, Methods and Applications*. *February 1996*

[2] Alfred Ka Yiu WOng, Pradeep Ray, N. Parameswaran and John Strassner *Ontology Mapping for the Interoperability Problem in Network Management*. IEEE Journal on selected areas in Communications, Vol 23, NO. 10. October 2005

[3] Jorge E. Lopez, Victor A. Villagra, Juan I. Asenio and Julio Berrocal. *Ontologies: Giving Semantics to Network Management Models*. IEEE network, special issue on Network Management, vol. 17 no. 3. May/June 2005

[4] Jorge E. Lopez, Victor and Julio Berrocal. *An ontology-based method to merge and map management information models*. Proceedings of the HP Openview University Association Ninth Plenary Workshop, Geneva, Switzerland.. July 2003

[5] Dejing Dou, Drew McDermott and Peishen Qi. *Ontology Translation on the Semantic Web*. Yale Computer Science department.

[6] Katharina Wolter, Thorsten Krebs and Lothar Hotz. *Ontology-based Model Comparison*. *HITeC e.V. c/o University of Hamburg*. *http://pi.informatik.uni-siegen.de/gi/fg211/VVUM07/pp/Wolter_Krebs_Hotz_2007_ppVVUM07.pdf* Last Retrieved May 4, 2007

[7] Mike lewis. *Understanding the Registry*. May 1999.

[8] Emre Kycyman and Yi-MinWang. *Discovering Correctness Constraints for Self-Management of System Configuration. Proceedings of the International Conference on Autonomic Computing (ICAC04).* 2004

[9] Mark Burgess and leen Frisch. *Promise and Cfengine: A working specification for cfengine 3.* November 29, 2005.

[10] Natalya F. Noy and Deborah L. McGuinness. *Ontology Development 101: A Guide to Creating Your First Ontology*.

[11] Thomas R. Gruber. *A Translation Approach to Portable Ontology Specifications. September 1992, Revised April 1993*

[12] *http://www.epistemics.co.uk/Notes/90-0-0.htm. Last retrieved April 25th 2007*

[13] John Strassner. *Knowledge Engineering using ontologies.*

[14] *Protégé* home Website. *http://protege.stanford.edu/overview/.* Last retrieved May 4th 2007

[15] Natalya F. Noy and Mark A. Musen. *Anchor-PROMPT: Using Non-Local Context for Semantic Matching.* Stanford Medical Informatics, Stanford University, Stanford, CA 94305-5479

[16] R.Mizoguchi, J. Vanweelkenhuysen, M. Ikeda. *Task Ontology for reuse of Problem Solving Knowledge.* Procedings of 2nd International Conference on Very Large-Scale Knowledge Bases, Tnschede, The Netherland (1995)

[17] Matthew Horridge, Holger Knublauch, Alan Rector, Robert Stevens, Chris Wroe. *A Pratical Guide To Building OWL Ontologies Using The Protege-OWL Plugin and CO-ODE Tools Edition 1.0.* The University of Manchester, August 27, 2004

[18] Aldo Gangemi, Carola Catenacci, Massimiliano Ciaramita and Jos Lehmann. *A theoretical framework for ontology evaluation and validation.* Laboratory for Applied Ontology, ISTC-CNR, Roma (Italy), *2005*

[19] Riichiro Mizoguchi and Mitsuru IKEDA. *Towards Ontology Engineering.* The Institute of Scientific and industrial Research, Osaka University, 567 Japan. Technical Report AI-TR-96-1, *1996*

[20] Salim K. Semy, Kevin N. Hetherington-Young, Steven E. Frey. *Ontology Engieenring: An Application perspective.* The MITRE Corporation

[21] Marc Ehrig and York Sure. *Ontology Mapping - An Integrated Approach.* Institut fr Angewandte Informatik und Formale Beschreibungsverfahren, Universitt Karlsruhe (TH), April 2004

[22] O. Lassila, D. McGuinness. *The Role of Frame-Based Representation on the Semantic Web.* Technical Report KSL-01-02, Knowledge Systems Laboratory, Stanford University

[23] Stephen Cranefield and Martin Purvis. *UML as an Ontology Modeling Language.* In the proceedings of the IJCAI-99 Workshop on Intelligent Information Integration

[24] Stephen Cranefield and Martin Purvis. *Applying The web Ontology Language to the Management Information Definitions. IEEE Communication Magazine, pages 68-74. July 2004.*

[25] Hans Hedbom, Stefan Lindskog, Stefan Axelsson and Erland Jonsson. *A Comparison of the Security of Windows NT and UNIX.* Presented at the Third Nordic Workshop on Secure IT Systems, NORDSEC98, 5-6 November, 1998, Trondheim, Norway. March 2, 1999

[26] J. Martin Serrano, Joan Serrat and John Strassner *Ontology for the Integration of Context in Network Management Operations and Business Support Systems.* IEEE Communication Magazine, Network and Service Management series

[27] Loom Project Home Page *http://www.isi.edu/isd/LOOM/*. Last retrieved May 16th 2007

[28] Michael Kifer, Georg Lausen and James Wu. *Journal of the Association for Computing Machinery (ACM),* May 1995

[29] Riichiro Mizoguchi. *Part 2: Ontology development, tools and languages*. *http://www.ei.sanken.osaka-u.ac.jp/pub/miz/Part2V3.pdf*. Last retrieved May 16th, 2007

[30] Clabby Analytics. *http://www.mkssoftware.com/docs/wp/wp_developersreport.pdf*. Last retrieved May 20, 2007

[31] Racer system Home Website. *http://www.racer-systems.com/products/racerpro/index.phtml/* Last retrieved May 16, 2007

[32] *http://www.dmtf.org/standards/stackmap/* Last retrieved May 16, 2007

[33] *http://www.wbemsolutions.com/tutorials/CIM/cimtutorial.pdf/* Last retrieved May 16, 2007

[34] *http://www.dmtf.org/about/faq/wbem/* Last retrieved May 16, 2007

[35] Konrad Rzeszutek *System Management using WBEM. http://sblim.sourceforge.net/doc/LWE-2005-02-WBEMSystemMgmt.pdf.* Last retrieved May 20, 2007.

[36] Microsoft Support Website. *http://support.microsoft.com/kb/q217098/*. Last retrieved May 18, 2007

[37] the Object Management Group, Inc.(OMG) *Common Object Request Broker Architecture for embedded CORBA. Draft Adopted Specification ptc/06-05-01,* May 2006

[38] Entreprise Management Associate (EMA). *Get the truth on Linux Management,* February 2006

[39] P. Emerald Chung, Yennun Huang, Shalini Yajnik,Deron Liang, Joanne C. Shih, Chung-Yih Wang and Yi-Min Wang. *DCOM and CORBA Side by Side, Step by Step, and Layer by Layer.* September 3, 1997

Appendix A

Appendices

A.1 Inferred taxonomy for the Windows *ntbackup* utility ontology generated in *Protégé*

A.2 Inferred taxonomy for Linux *tar* program ontology generated in *Protégé*

A.3 RDF/XML code generated with *Protégé* for the Windows *ntbackup* utility Ontology

```
<?xml version="1.0"?>

<!DOCTYPE rdf:RDF [
<!ENTITY owl "http://www.w3.org/2002/07/owl#" >
<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
<!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
<!ENTITY p1 "http://www.owl-ontologies.com/assert.owl#" >
<!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>

<rdf:RDF xmlns="http://protege.stanford.edu/plugins/owl/protege#"
xml:base="http://protege.stanford.edu/plugins/owl/protege"
xmlns:p1="http://www.owl-ontologies.com/assert.owl#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:owl="http://www.w3.org/2002/07/owl#">
<owl:Ontology rdf:about="">
<owl:imports rdf:resource="http://protege.stanford.edu/plugins/owl/
protege"/>
</owl:Ontology>
<owl:AllDifferent>
<owl:distinctMembers rdf:parseType="Collection"/>
</owl:AllDifferent>
<appendSwitch rdf:ID="A"/>
<owl:Class rdf:ID="accessRestrictionSwitch">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#hasSwitchValue"/>
<owl:allValuesFrom rdf:resource="#switchValue1"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="#switch"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#precede"/>
<owl:allValuesFrom rdf:resource="#switchValue1"/>
</owl:Restriction>
</rdfs:subClassOf>
<owl:disjointWith rdf:resource="#logFileTypeSwitch"/>
<owl:disjointWith rdf:resource="#dataVerificationSwitch"/>
<owl:disjointWith rdf:resource="#appendSwitch"/>
<owl:disjointWith rdf:resource="#hardwareCompressionSwitch"/>
```

```
<owl:disjointWith rdf:resource="#systemstateSwitch"/>
<owl:disjointWith rdf:resource="#backupTypeSwitch"/>
<owl:disjointWith rdf:resource="#tapeNameSwitch"/>
<owl:disjointWith rdf:resource="#mediaNameSwitch"/>
<owl:disjointWith rdf:resource="#fileNameSwitch"/>
<owl:disjointWith rdf:resource="#poolNameSwitch"/>
<owl:disjointWith rdf:resource="#jobNameSwitch"/>
<rdfs:comment rdf:datatype="&xsd;string"
>Restricts access to this tape to the owner or members of the
Administrators group</rdfs:comment>
</owl:Class>
<owl:Class rdf:ID="appendSwitch">
<rdfs:subClassOf rdf:resource="#switch"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#NotCompatibleWith"/>
<owl:hasValue rdf:resource="#P"/>
</owl:Restriction>
</rdfs:subClassOf>
<owl:disjointWith rdf:resource="#fileNameSwitch"/>
<owl:disjointWith rdf:resource="#backupTypeSwitch"/>
<owl:disjointWith rdf:resource="#logFileTypeSwitch"/>
<owl:disjointWith rdf:resource="#systemstateSwitch"/>
<owl:disjointWith rdf:resource="#jobNameSwitch"/>
<owl:disjointWith rdf:resource="#poolNameSwitch"/>
<owl:disjointWith rdf:resource="#accessRestrictionSwitch"/>
<owl:disjointWith rdf:resource="#dataVerificationSwitch"/>
<owl:disjointWith rdf:resource="#hardwareCompressionSwitch"/>
<owl:disjointWith rdf:resource="#tapeNameSwitch"/>
<owl:disjointWith rdf:resource="#mediaNameSwitch"/>
</owl:Class>
<command2 rdf:ID="backup"/>
<owl:Class rdf:ID="BackupSetDescription">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#isLabelOf"/>
<owl:allValuesFrom rdf:resource="#dataStorage"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="#winBackupOperation"/>
<owl:disjointWith rdf:resource="#fileStorage"/>
<owl:disjointWith rdf:resource="#command"/>
<owl:disjointWith rdf:resource="#dataToBckp"/>
<owl:disjointWith rdf:resource="#switchValue"/>
<owl:disjointWith rdf:resource="#jobName"/>
<owl:disjointWith rdf:resource="#tapeStorageName"/>
<owl:disjointWith rdf:resource="#switch"/>
```

```
<owl:disjointWith rdf:resource="#dataStorage"/>
</owl:Class>
<owl:Class rdf:ID="backupTypeSwitch">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#hasSwitchValue"/>
<owl:allValuesFrom rdf:resource="#backupTypeSwitchValue"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#precede"/>
<owl:allValuesFrom rdf:resource="#backupTypeSwitchValue"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="#switch"/>
<owl:disjointWith rdf:resource="#accessRestrictionSwitch"/>
<owl:disjointWith rdf:resource="#dataVerificationSwitch"/>
<owl:disjointWith rdf:resource="#poolNameSwitch"/>
<owl:disjointWith rdf:resource="#jobNameSwitch"/>
<owl:disjointWith rdf:resource="#appendSwitch"/>
<owl:disjointWith rdf:resource="#fileNameSwitch"/>
<owl:disjointWith rdf:resource="#tapeNameSwitch"/>
<owl:disjointWith rdf:resource="#systemstateSwitch"/>
<owl:disjointWith rdf:resource="#logFileTypeSwitch"/>
<owl:disjointWith rdf:resource="#hardwareCompressionSwitch"/>
<owl:disjointWith rdf:resource="#mediaNameSwitch"/>
<rdfs:comment rdf:datatype="&xsd;string"
>Specifies the backup type. It must be one of the following:
normal, copy, differential, incremental, or daily</rdfs:comment>
</owl:Class>
<owl:Class rdf:ID="backupTypeSwitchValue">
<rdfs:subClassOf rdf:resource="#switchValue"/>
<owl:disjointWith rdf:resource="#switchValue3"/>
<owl:disjointWith rdf:resource="#switchValue2"/>
<owl:disjointWith rdf:resource="#switchValue1"/>
</owl:Class>
<owl:Class rdf:ID="command">
<rdfs:subClassOf rdf:resource="#winBackupOperation"/>
<owl:disjointWith rdf:resource="#BackupSetDescription"/>
<owl:disjointWith rdf:resource="#dataStorage"/>
<owl:disjointWith rdf:resource="#dataToBckp"/>
<owl:disjointWith rdf:resource="#jobName"/>
<owl:disjointWith rdf:resource="#switch"/>
<owl:disjointWith rdf:resource="#switchValue"/>
</owl:Class>
<owl:Class rdf:ID="command1">
```

```
<rdfs:subClassOf rdf:resource="#command"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#precede"/>
<owl:hasValue rdf:resource="#backup"/>
</owl:Restriction>
</rdfs:subClassOf>
<owl:disjointWith rdf:resource="#command2"/>
</owl:Class>
<owl:Class rdf:ID="command2">
<rdfs:subClassOf rdf:resource="#command"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#follow"/>
<owl:hasValue rdf:resource="#ntbackup"/>
</owl:Restriction>
</rdfs:subClassOf>
<owl:disjointWith rdf:resource="#command1"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="compatibleWith">
<rdfs:domain rdf:resource="#switch"/>
<rdfs:range rdf:resource="#switch"/>
</owl:ObjectProperty>
<backupTypeSwitchValue rdf:ID="copy"/>
<backupTypeSwitchValue rdf:ID="daily"/>
<owl:Class rdf:ID="dataStorage">
<rdfs:subClassOf rdf:resource="#winBackupOperation"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#hasLabel"/>
<owl:allValuesFrom rdf:resource="#BackupSetDescription"/>
</owl:Restriction>
</rdfs:subClassOf>
<owl:disjointWith rdf:resource="#dataToBckp"/>
<owl:disjointWith rdf:resource="#BackupSetDescription"/>
<owl:disjointWith rdf:resource="#command"/>
<owl:disjointWith rdf:resource="#jobName"/>
<owl:disjointWith rdf:resource="#switch"/>
<owl:disjointWith rdf:resource="#switchValue"/>
</owl:Class>
<owl:Class rdf:ID="dataToBckp">
<rdfs:subClassOf rdf:resource="#winBackupOperation"/>
<owl:disjointWith rdf:resource="#dataStorage"/>
<owl:disjointWith rdf:resource="#BackupSetDescription"/>
<owl:disjointWith rdf:resource="#command"/>
<owl:disjointWith rdf:resource="#jobName"/>
<owl:disjointWith rdf:resource="#switch"/>
```

```
<owl:disjointWith rdf:resource="#switchValue"/>
</owl:Class>
<owl:Class rdf:ID="dataVerificationSwitch">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#hasSwitchValue"/>
<owl:allValuesFrom rdf:resource="#switchValue1"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#precede"/>
<owl:allValuesFrom rdf:resource="#switchValue1"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="#switch"/>
<owl:disjointWith rdf:resource="#poolNameSwitch"/>
<owl:disjointWith rdf:resource="#systemstateSwitch"/>
<owl:disjointWith rdf:resource="#mediaNameSwitch"/>
<owl:disjointWith rdf:resource="#appendSwitch"/>
<owl:disjointWith rdf:resource="#backupTypeSwitch"/>
<owl:disjointWith rdf:resource="#jobNameSwitch"/>
<owl:disjointWith rdf:resource="#accessRestrictionSwitch"/>
<owl:disjointWith rdf:resource="#hardwareCompressionSwitch"/>
<owl:disjointWith rdf:resource="#fileNameSwitch"/>
<owl:disjointWith rdf:resource="#logFileTypeSwitch"/>
<owl:disjointWith rdf:resource="#tapeNameSwitch"/>
<rdfs:comment rdf:datatype="&xsd;string"
>Verifies the data after the backup is complete.</rdfs:comment>
</owl:Class>
<backupTypeSwitchValue rdf:ID="differential"/>
<owl:Class rdf:ID="directoryPath">
<rdfs:subClassOf rdf:resource="#fileStorage"/>
<owl:disjointWith rdf:resource="#fileName"/>
</owl:Class>
<fileNameSwitch rdf:ID="F"/>
<switchValue3 rdf:ID="f"/>
<owl:Class rdf:ID="fileListName">
<rdfs:subClassOf rdf:resource="#dataToBckp"/>
<owl:disjointWith rdf:resource="#wdirectoryPath"/>
<owl:disjointWith rdf:resource="#wfileName"/>
</owl:Class>
<owl:Class rdf:ID="fileName">
<rdfs:subClassOf rdf:resource="#fileStorage"/>
<owl:disjointWith rdf:resource="#directoryPath"/>
</owl:Class>
<owl:Class rdf:ID="fileNameSwitch">
```

```
<rdfs:subClassOf rdf:resource="#switch"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#NotCompatibleWith"/>
<owl:hasValue rdf:resource="#T"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#isSwitchOf"/>
<owl:allValuesFrom rdf:resource="#fileName"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#NotCompatibleWith"/>
<owl:hasValue rdf:resource="#J"/>
</owl:Restriction>
</rdfs:subClassOf>
<owl:disjointWith rdf:resource="#mediaNameSwitch"/>
<owl:disjointWith rdf:resource="#tapeNameSwitch"/>
<owl:disjointWith rdf:resource="#appendSwitch"/>
<owl:disjointWith rdf:resource="#poolNameSwitch"/>
<owl:disjointWith rdf:resource="#accessRestrictionSwitch"/>
<owl:disjointWith rdf:resource="#backupTypeSwitch"/>
<owl:disjointWith rdf:resource="#logFileTypeSwitch"/>
<owl:disjointWith rdf:resource="#jobNameSwitch"/>
<owl:disjointWith rdf:resource="#systemstateSwitch"/>
<owl:disjointWith rdf:resource="#hardwareCompressionSwitch"/>
<owl:disjointWith rdf:resource="#dataVerificationSwitch"/>
</owl:Class>
<owl:Class rdf:ID="fileStorage">
<rdfs:subClassOf rdf:resource="#dataStorage"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#hasSwitch"/>
<owl:hasValue rdf:resource="#F"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#follow"/>
<owl:hasValue rdf:resource="#F"/>
</owl:Restriction>
</rdfs:subClassOf>
<owl:disjointWith rdf:resource="#BackupSetDescription"/>
<owl:disjointWith rdf:resource="#tapeStorageName"/>
```

```
</owl:Class>
<owl:ObjectProperty rdf:ID="follow">
<owl:inverseOf rdf:resource="#precede"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="hardwareCompressionSwitch">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#hasSwitchValue"/>
<owl:allValuesFrom rdf:resource="#switchValue2"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="#switch"/>
<owl:disjointWith rdf:resource="#appendSwitch"/>
<owl:disjointWith rdf:resource="#accessRestrictionSwitch"/>
<owl:disjointWith rdf:resource="#backupTypeSwitch"/>
<owl:disjointWith rdf:resource="#tapeNameSwitch"/>
<owl:disjointWith rdf:resource="#poolNameSwitch"/>
<owl:disjointWith rdf:resource="#mediaNameSwitch"/>
<owl:disjointWith rdf:resource="#jobNameSwitch"/>
<owl:disjointWith rdf:resource="#dataVerificationSwitch"/>
<owl:disjointWith rdf:resource="#logFileTypeSwitch"/>
<owl:disjointWith rdf:resource="#systemstateSwitch"/>
<owl:disjointWith rdf:resource="#fileNameSwitch"/>
<rdfs:comment rdf:datatype="&xsd;string"
>Uses hardware compression, if available, on the tape drive.
</rdfs:comment>
</owl:Class>
<owl:ObjectProperty rdf:ID="hasEndTime"/>
<owl:ObjectProperty rdf:ID="hasFrequency"/>
<owl:ObjectProperty rdf:ID="hasLabel">
<owl:inverseOf rdf:resource="#isLabelOf"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasName"/>
<owl:ObjectProperty rdf:ID="hasPath">
<rdf:type rdf:resource="&owl;FunctionalProperty"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasSize"/>
<owl:ObjectProperty rdf:ID="hasStartingTime"/>
<owl:ObjectProperty rdf:ID="hasSwitch">
<rdf:type rdf:resource="&owl;FunctionalProperty"/>
<rdfs:domain>
<owl:Class>
<owl:unionOf rdf:parseType="Collection">
<owl:Class rdf:about="#dataStorage"/>
<owl:Class rdf:about="#dataToBckp"/>
<owl:Class rdf:about="#jobName"/>
</owl:unionOf>
```

```
</owl:Class>
</rdfs:domain>
<rdfs:range rdf:resource="#switch"/>
<owl:inverseOf rdf:resource="#isSwitchOf"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasSwitchValue">
<owl:inverseOf rdf:resource="#isSwitchValue"/>
</owl:ObjectProperty>
<hardwareCompressionSwitch rdf:ID="HC"/>
<backupTypeSwitchValue rdf:ID="incremental"/>
<owl:ObjectProperty rdf:ID="isLabelOf">
<owl:inverseOf rdf:resource="#hasLabel"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isSwitchOf">
<rdf:type rdf:resource="&owl;InverseFunctionalProperty"/>
<rdfs:domain rdf:resource="#switch"/>
<rdfs:range>
<owl:Class>
<owl:unionOf rdf:parseType="Collection">
<owl:Class rdf:about="#dataStorage"/>
<owl:Class rdf:about="#dataToBckp"/>
<owl:Class rdf:about="#jobName"/>
</owl:unionOf>
</owl:Class>
</rdfs:range>
<owl:inverseOf rdf:resource="#hasSwitch"/>
<rdfs:comment rdf:datatype="&xsd;string"></rdfs:comment>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isSwitchValue">
<owl:inverseOf rdf:resource="#hasSwitchValue"/>
</owl:ObjectProperty>
<jobNameSwitch rdf:ID="J"/>
<owl:Class rdf:ID="jobName">
<rdfs:subClassOf rdf:resource="#winBackupOperation"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#hasSwitch"/>
<owl:hasValue rdf:resource="#J"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#follow"/>
<owl:hasValue rdf:resource="#J"/>
</owl:Restriction>
</rdfs:subClassOf>
<owl:disjointWith rdf:resource="#dataToBckp"/>
```

```
<owl:disjointWith rdf:resource="#BackupSetDescription"/>
<owl:disjointWith rdf:resource="#switch"/>
<owl:disjointWith rdf:resource="#command"/>
<owl:disjointWith rdf:resource="#dataStorage"/>
<owl:disjointWith rdf:resource="#switchValue"/>
<rdfs:comment rdf:datatype="&xsd;string"
>name of the job as written in the log file</rdfs:comment>
</owl:Class>
<owl:Class rdf:ID="jobNameSwitch">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#isSwitchOf"/>
<owl:allValuesFrom rdf:resource="#jobName"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="#switch"/>
<owl:disjointWith rdf:resource="#backupTypeSwitch"/>
<owl:disjointWith rdf:resource="#mediaNameSwitch"/>
<owl:disjointWith rdf:resource="#accessRestrictionSwitch"/>
<owl:disjointWith rdf:resource="#tapeNameSwitch"/>
<owl:disjointWith rdf:resource="#poolNameSwitch"/>
<owl:disjointWith rdf:resource="#appendSwitch"/>
<owl:disjointWith rdf:resource="#logFileTypeSwitch"/>
<owl:disjointWith rdf:resource="#systemstateSwitch"/>
<owl:disjointWith rdf:resource="#dataVerificationSwitch"/>
<owl:disjointWith rdf:resource="#fileNameSwitch"/>
<owl:disjointWith rdf:resource="#hardwareCompressionSwitch"/>
<rdfs:comment rdf:datatype="&xsd;string"
>Specifies the job name to be used in the log file The job name
usually describes the files and folders you are backing up in
the current backup job as well as the date and time you backed up
 the files.</rdfs:comment>
</owl:Class>
<logFileTypeSwitch rdf:ID="L"/>
<switchValue3 rdf:ID="l"/>
<owl:Class rdf:ID="localPath">
<rdfs:subClassOf rdf:resource="#directoryPath"/>
<owl:disjointWith rdf:resource="#UNCNetParh"/>
</owl:Class>
<owl:Class rdf:ID="logFileTypeSwitch">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#hasSwitchValue"/>
<owl:allValuesFrom rdf:resource="#switchValue3"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="#switch"/>
```

```
<owl:disjointWith rdf:resource="#jobNameSwitch"/>
<owl:disjointWith rdf:resource="#appendSwitch"/>
<owl:disjointWith rdf:resource="#hardwareCompressionSwitch"/>
<owl:disjointWith rdf:resource="#fileNameSwitch"/>
<owl:disjointWith rdf:resource="#poolNameSwitch"/>
<owl:disjointWith rdf:resource="#systemstateSwitch"/>
<owl:disjointWith rdf:resource="#dataVerificationSwitch"/>
<owl:disjointWith rdf:resource="#accessRestrictionSwitch"/>
<owl:disjointWith rdf:resource="#tapeNameSwitch"/>
<owl:disjointWith rdf:resource="#backupTypeSwitch"/>
<owl:disjointWith rdf:resource="#mediaNameSwitch"/>
<rdfs:comment rdf:datatype="&xsd;string"
>Specifies the type of log file: f=full, s=summary, n=none
(no log file is created).</rdfs:comment>
</owl:Class>
<backupTypeSwitch rdf:ID="M"/>
<owl:Class rdf:ID="mediaName">
<rdfs:subClassOf rdf:resource="#tapeStorageName"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#hasSwitch"/>
<owl:hasValue rdf:resource="#N"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#follow"/>
<owl:hasValue rdf:resource="#N"/>
</owl:Restriction>
</rdfs:subClassOf>
<owl:disjointWith rdf:resource="#tapeName"/>
</owl:Class>
<owl:Class rdf:ID="mediaNameSwitch">
<rdfs:subClassOf rdf:resource="#switch"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#isSwitchOf"/>
<owl:allValuesFrom rdf:resource="#mediaName"/>
</owl:Restriction>
</rdfs:subClassOf>
<owl:disjointWith rdf:resource="#dataVerificationSwitch"/>
<owl:disjointWith rdf:resource="#appendSwitch"/>
<owl:disjointWith rdf:resource="#tapeNameSwitch"/>
<owl:disjointWith rdf:resource="#jobNameSwitch"/>
<owl:disjointWith rdf:resource="#poolNameSwitch"/>
<owl:disjointWith rdf:resource="#logFileTypeSwitch"/>
<owl:disjointWith rdf:resource="#accessRestrictionSwitch"/>
```

```
<owl:disjointWith rdf:resource="#fileNameSwitch"/>
<owl:disjointWith rdf:resource="#hardwareCompressionSwitch"/>
<owl:disjointWith rdf:resource="#backupTypeSwitch"/>
<owl:disjointWith rdf:resource="#systemstateSwitch"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="MustBeUsedWith"/>
<mediaNameSwitch rdf:ID="N"/>
<switchValue3 rdf:ID="n"/>
<switchValue1 rdf:ID="no"/>
<backupTypeSwitchValue rdf:ID="normal"/>
<owl:ObjectProperty rdf:ID="NotCompatibleWith">
<rdf:type rdf:resource="&owl;SymmetricProperty"/>
<rdf:type rdf:resource="&owl;TransitiveProperty"/>
<rdfs:domain rdf:resource="#switch"/>
<rdfs:range rdf:resource="#switch"/>
<owl:inverseOf rdf:resource="#NotCompatibleWith"/>
</owl:ObjectProperty>
<command1 rdf:ID="ntbackup"/>
<switchValue2 rdf:ID="off"/>
<switchValue2 rdf:ID="on"/>
<poolNameSwitch rdf:ID="P"/>
<owl:Class rdf:ID="poolNameSwitch">
<rdfs:subClassOf rdf:resource="#switch"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#NotCompatibleWith"/>
<owl:hasValue rdf:resource="#F"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#NotCompatibleWith"/>
<owl:hasValue rdf:resource="#A"/>
</owl:Restriction>
</rdfs:subClassOf>
<owl:disjointWith rdf:resource="#mediaNameSwitch"/>
<owl:disjointWith rdf:resource="#logFileTypeSwitch"/>
<owl:disjointWith rdf:resource="#tapeNameSwitch"/>
<owl:disjointWith rdf:resource="#jobNameSwitch"/>
<owl:disjointWith rdf:resource="#systemstateSwitch"/>
<owl:disjointWith rdf:resource="#backupTypeSwitch"/>
<owl:disjointWith rdf:resource="#fileNameSwitch"/>
<owl:disjointWith rdf:resource="#dataVerificationSwitch"/>
<owl:disjointWith rdf:resource="#appendSwitch"/>
<owl:disjointWith rdf:resource="#accessRestrictionSwitch"/>
<owl:disjointWith rdf:resource="#hardwareCompressionSwitch"/>
</owl:Class>
```

```
<owl:ObjectProperty rdf:ID="precede">
<owl:inverseOf rdf:resource="#follow"/>
</owl:ObjectProperty>
<accessRestrictionSwitch rdf:ID="R"/>
<owl:Class rdf:ID="switch">
<rdfs:subClassOf rdf:resource="#winBackupOperation"/>
<owl:disjointWith rdf:resource="#BackupSetDescription"/>
<owl:disjointWith rdf:resource="#command"/>
<owl:disjointWith rdf:resource="#dataStorage"/>
<owl:disjointWith rdf:resource="#dataToBckp"/>
<owl:disjointWith rdf:resource="#jobName"/>
<owl:disjointWith rdf:resource="#switchValue"/>
</owl:Class>
<owl:Class rdf:ID="switchValue">
<rdfs:subClassOf rdf:resource="#winBackupOperation"/>
<owl:disjointWith rdf:resource="#BackupSetDescription"/>
<owl:disjointWith rdf:resource="#command"/>
<owl:disjointWith rdf:resource="#dataStorage"/>
<owl:disjointWith rdf:resource="#dataToBckp"/>
<owl:disjointWith rdf:resource="#jobName"/>
<owl:disjointWith rdf:resource="#switch"/>
<rdfs:comment rdf:datatype="&xsd;string"
>some values from specific switches : /RS:{yes|no}
and HC:{on|off}</rdfs:comment>
</owl:Class>
<owl:Class rdf:ID="switchValue1">
<rdfs:subClassOf rdf:resource="#switchValue"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#isSwitchValue"/>
<owl:someValuesFrom rdf:resource="#accessRestrictionSwitch"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#isSwitchValue"/>
<owl:someValuesFrom rdf:resource="#dataVerificationSwitch"/>
</owl:Restriction>
</rdfs:subClassOf>
<owl:disjointWith rdf:resource="#switchValue3"/>
<owl:disjointWith rdf:resource="#switchValue2"/>
<owl:disjointWith rdf:resource="#backupTypeSwitchValue"/>
</owl:Class>
<owl:Class rdf:ID="switchValue2">
<rdfs:subClassOf rdf:resource="#switchValue"/>
<owl:disjointWith rdf:resource="#switchValue3"/>
<owl:disjointWith rdf:resource="#switchValue1"/>
```

```
<owl:disjointWith rdf:resource="#backupTypeSwitchValue"/>
</owl:Class>
<owl:Class rdf:ID="switchValue3">
<rdfs:subClassOf rdf:resource="#switchValue"/>
<owl:disjointWith rdf:resource="#switchValue2"/>
<owl:disjointWith rdf:resource="#switchValue1"/>
<owl:disjointWith rdf:resource="#backupTypeSwitchValue"/>
</owl:Class>
<owl:Class rdf:ID="systemstateSwitch">
<rdfs:subClassOf rdf:resource="#switch"/>
<owl:disjointWith rdf:resource="#dataVerificationSwitch"/>
<owl:disjointWith rdf:resource="#fileNameSwitch"/>
<owl:disjointWith rdf:resource="#hardwareCompressionSwitch"/>
<owl:disjointWith rdf:resource="#appendSwitch"/>
<owl:disjointWith rdf:resource="#poolNameSwitch"/>
<owl:disjointWith rdf:resource="#jobNameSwitch"/>
<owl:disjointWith rdf:resource="#accessRestrictionSwitch"/>
<owl:disjointWith rdf:resource="#logFileTypeSwitch"/>
<owl:disjointWith rdf:resource="#backupTypeSwitch"/>
<owl:disjointWith rdf:resource="#mediaNameSwitch"/>
<owl:disjointWith rdf:resource="#tapeNameSwitch"/>
</owl:Class>
<tapeNameSwitch rdf:ID="T"/>
<owl:Class rdf:ID="tapeName">
<rdfs:subClassOf rdf:resource="#tapeStorageName"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#hasSwitch"/>
<owl:hasValue rdf:resource="#T"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#follow"/>
<owl:hasValue rdf:resource="#T"/>
</owl:Restriction>
</rdfs:subClassOf>
<owl:disjointWith rdf:resource="#mediaName"/>
</owl:Class>
<owl:Class rdf:ID="tapeNameSwitch">
<rdfs:subClassOf rdf:resource="#switch"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#hasSwitch"/>
<owl:allValuesFrom rdf:resource="#tapeName"/>
</owl:Restriction>
</rdfs:subClassOf>
```

```
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#NotCompatibleWith"/>
<owl:hasValue rdf:resource="#P"/>
</owl:Restriction>
</rdfs:subClassOf>
<owl:disjointWith rdf:resource="#dataVerificationSwitch"/>
<owl:disjointWith rdf:resource="#fileNameSwitch"/>
<owl:disjointWith rdf:resource="#mediaNameSwitch"/>
<owl:disjointWith rdf:resource="#jobNameSwitch"/>
<owl:disjointWith rdf:resource="#logFileTypeSwitch"/>
<owl:disjointWith rdf:resource="#hardwareCompressionSwitch"/>
<owl:disjointWith rdf:resource="#poolNameSwitch"/>
<owl:disjointWith rdf:resource="#backupTypeSwitch"/>
<owl:disjointWith rdf:resource="#accessRestrictionSwitch"/>
<owl:disjointWith rdf:resource="#appendSwitch"/>
<owl:disjointWith rdf:resource="#systemstateSwitch"/>
</owl:Class>
<owl:Class rdf:ID="tapeStorageName">
<rdfs:subClassOf rdf:resource="#dataStorage"/>
<owl:disjointWith rdf:resource="#BackupSetDescription"/>
<owl:disjointWith rdf:resource="#fileStorage"/>
</owl:Class>
<owl:Class rdf:ID="UNCNetParh">
<rdfs:subClassOf rdf:resource="#directoryPath"/>
<owl:disjointWith rdf:resource="#localPath"/>
</owl:Class>
<dataVerificationSwitch rdf:ID="V"/>
<owl:Class rdf:ID="wdirectoryPath">
<rdfs:subClassOf rdf:resource="#dataToBckp"/>
<owl:disjointWith rdf:resource="#fileListName"/>
<owl:disjointWith rdf:resource="#wfileName"/>
</owl:Class>
<owl:Class rdf:ID="wfileName">
<rdfs:subClassOf rdf:resource="#dataToBckp"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#hasPath"/>
<owl:cardinality rdf:datatype="&xsd;int">1</owl:cardinality>
</owl:Restriction>
</rdfs:subClassOf>
<owl:disjointWith rdf:resource="#fileListName"/>
<owl:disjointWith rdf:resource="#wdirectoryPath"/>
</owl:Class>
<owl:Class rdf:ID="winBackupOperation"/>
<switchValue1 rdf:ID="yes"/>
</rdf:RDF>
```

A.4 RDF/XML code generated with *Protégé* for the *tar* Linux backup ontology

```
<?xml version="1.0"?>


<!DOCTYPE rdf:RDF [
<!ENTITY owl "http://www.w3.org/2002/07/owl#" >
<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
<!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
<!ENTITY p1 "http://www.owl-ontologies.com/assert.owl#" >
<!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>

<rdf:RDF xmlns="http://www.owl-ontologies.com/Ontology1177518345.owl#"
xml:base="http://www.owl-ontologies.com/Ontology1177518345.owl"
xmlns:p1="http://www.owl-ontologies.com/assert.owl#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:owl="http://www.w3.org/2002/07/owl#">
<owl:Ontology rdf:about=""/>
<appendParam rdf:ID="A"/>
<owl:Class rdf:ID="appendParam">
<rdfs:subClassOf rdf:resource="#functions"/>
<owl:disjointWith rdf:resource="#appendToEndOfStorage"/>
<owl:disjointWith rdf:resource="#createArchiveParam"/>
<owl:disjointWith rdf:resource="#updateParam"/>
<rdfs:comment rdf:datatype="&xsd;string"
>append tar files to an archive</rdfs:comment>
</owl:Class>
<owl:Class rdf:ID="appendToEndOfStorage">
<rdfs:subClassOf rdf:resource="#functions"/>
<owl:disjointWith rdf:resource="#appendParam"/>
<owl:disjointWith rdf:resource="#createArchiveParam"/>
<owl:disjointWith rdf:resource="#updateParam"/>
<rdfs:comment rdf:datatype="&xsd;string"
>append files to the end of an archive</rdfs:comment>
</owl:Class>
<owl:Class rdf:ID="archiveName">
<rdfs:subClassOf rdf:resource="#storage"/>
<owl:disjointWith rdf:resource="#tapeFileName"/>
<owl:disjointWith rdf:resource="#archivePath"/>
</owl:Class>
<owl:Class rdf:ID="archiveNameParam">
<rdfs:subClassOf>
```

```
<owl:Restriction>
<owl:onProperty rdf:resource="#isParamOf"/>
<owl:allValuesFrom rdf:resource="#storage"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="#options"/>
<owl:disjointWith rdf:resource="#preserveAccessParam"/>
<owl:disjointWith rdf:resource="#typeOfbackup"/>
<owl:disjointWith rdf:resource="#permission"/>
<owl:disjointWith rdf:resource="#compressionParam"/>
<owl:disjointWith rdf:resource="#fileListParam"/>
<owl:disjointWith rdf:resource="#verificationParam"/>
<owl:disjointWith rdf:resource="#fileChangeLatestParam"/>
<owl:disjointWith rdf:resource="#keepOldFileParam"/>
<owl:disjointWith rdf:resource="#onlyStoreFileNewerthanParam"/>
<owl:disjointWith rdf:resource="#exclusionParam"/>
</owl:Class>
<owl:Class rdf:ID="archivePath">
<rdfs:subClassOf rdf:resource="#storage"/>
<owl:disjointWith rdf:resource="#tapeFileName"/>
<owl:disjointWith rdf:resource="#archiveName"/>
</owl:Class>
<preserveAccessParam rdf:ID="atime-preserve"/>
<createArchiveParam rdf:ID="c"/>
<owl:Class rdf:ID="command">
<rdfs:subClassOf rdf:resource="#linuxBckp_Operation"/>
<owl:disjointWith rdf:resource="#dataToBackup"/>
<owl:disjointWith rdf:resource="#functions"/>
<owl:disjointWith rdf:resource="#options"/>
<owl:disjointWith rdf:resource="#storage"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="compatibleWith">
<owl:inverseOf rdf:resource="#incompatibeWith"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="compressionParam">
<rdfs:subClassOf rdf:resource="#options"/>
<owl:disjointWith rdf:resource="#preserveAccessParam"/>
<owl:disjointWith rdf:resource="#exclusionParam"/>
<owl:disjointWith rdf:resource="#archiveNameParam"/>
<owl:disjointWith rdf:resource="#typeOfbackup"/>
<owl:disjointWith rdf:resource="#fileChangeLatestParam"/>
<owl:disjointWith rdf:resource="#fileListParam"/>
<owl:disjointWith rdf:resource="#verificationParam"/>
<owl:disjointWith rdf:resource="#keepOldFileParam"/>
<owl:disjointWith rdf:resource="#onlyStoreFileNewerthanParam"/>
<owl:disjointWith rdf:resource="#permission"/>
<rdfs:comment rdf:datatype="&xsd;string"></rdfs:comment>
```

```
</owl:Class>
<owl:Class rdf:ID="createArchiveParam">
<rdfs:subClassOf rdf:resource="#functions"/>
<owl:disjointWith rdf:resource="#appendParam"/>
<owl:disjointWith rdf:resource="#appendToEndOfStorage"/>
<owl:disjointWith rdf:resource="#updateParam"/>
<rdfs:comment rdf:datatype="&xsd;string"
>create a new Archive</rdfs:comment>
</owl:Class>
<owl:Class rdf:ID="dataToBackup">
<rdfs:subClassOf rdf:resource="#linuxBckp_Operation"/>
<owl:disjointWith rdf:resource="#command"/>
<owl:disjointWith rdf:resource="#functions"/>
<owl:disjointWith rdf:resource="#options"/>
<owl:disjointWith rdf:resource="#storage"/>
</owl:Class>
<owl:Class rdf:ID="directoryToBckpPath">
<rdfs:subClassOf rdf:resource="#dataToBackup"/>
<owl:disjointWith rdf:resource="#singlefileName"/>
<owl:disjointWith rdf:resource="#fileListName"/>
</owl:Class>
<owl:Class rdf:ID="exclusionFilePatern">
<owl:disjointWith rdf:resource="#time"/>
<owl:disjointWith rdf:resource="#linuxBckp_Operation"/>
</owl:Class>
<owl:Class rdf:ID="exclusionParam">
<rdfs:subClassOf rdf:resource="#options"/>
<owl:disjointWith rdf:resource="#preserveAccessParam"/>
<owl:disjointWith rdf:resource="#fileChangeLatestParam"/>
<owl:disjointWith rdf:resource="#verificationParam"/>
<owl:disjointWith rdf:resource="#keepOldFileParam"/>
<owl:disjointWith rdf:resource="#permission"/>
<owl:disjointWith rdf:resource="#typeOfbackup"/>
<owl:disjointWith rdf:resource="#compressionParam"/>
<owl:disjointWith rdf:resource="#archiveNameParam"/>
<owl:disjointWith rdf:resource="#fileListParam"/>
<owl:disjointWith rdf:resource="#onlyStoreFileNewerthanParam"/>
<rdfs:comment rdf:datatype="&xsd;string"
>exclude files matching patterns listed in FILE</rdfs:comment>
</owl:Class>
<archiveNameParam rdf:ID="f"/>
<owl:Class rdf:ID="fileChangeLatestParam">
<rdfs:subClassOf rdf:resource="#options"/>
<owl:disjointWith rdf:resource="#fileListParam"/>
<owl:disjointWith rdf:resource="#typeOfbackup"/>
<owl:disjointWith rdf:resource="#onlyStoreFileNewerthanParam"/>
<owl:disjointWith rdf:resource="#archiveNameParam"/>
```

```
<owl:disjointWith rdf:resource="#exclusionParam"/>
<owl:disjointWith rdf:resource="#preserveAccessParam"/>
<owl:disjointWith rdf:resource="#verificationParam"/>
<owl:disjointWith rdf:resource="#keepOldFileParam"/>
<owl:disjointWith rdf:resource="#permission"/>
<owl:disjointWith rdf:resource="#compressionParam"/>
<rdfs:comment rdf:datatype="&xsd;string"
>only store files whose contents have changed after DATE</rdfs:comment>
</owl:Class>
<owl:Class rdf:ID="fileChangeLatestTime">
<rdfs:subClassOf rdf:resource="#time"/>
<owl:disjointWith rdf:resource="#preserveAccessTime"/>
<owl:disjointWith rdf:resource="#onlyStoreFileNewerThanTime"/>
</owl:Class>
<owl:Class rdf:ID="fileListName">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#hasParam"/>
<owl:hasValue rdf:resource="#T"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="#dataToBackup"/>
<owl:disjointWith rdf:resource="#singlefileName"/>
<owl:disjointWith rdf:resource="#directoryToBckpPath"/>
</owl:Class>
<owl:Class rdf:ID="fileListParam">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#isParamOf"/>
<owl:allValuesFrom rdf:resource="#fileListName"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="#options"/>
<owl:disjointWith rdf:resource="#typeOfbackup"/>
<owl:disjointWith rdf:resource="#fileChangeLatestParam"/>
<owl:disjointWith rdf:resource="#onlyStoreFileNewerthanParam"/>
<owl:disjointWith rdf:resource="#verificationParam"/>
<owl:disjointWith rdf:resource="#permission"/>
<owl:disjointWith rdf:resource="#compressionParam"/>
<owl:disjointWith rdf:resource="#archiveNameParam"/>
<owl:disjointWith rdf:resource="#preserveAccessParam"/>
<owl:disjointWith rdf:resource="#keepOldFileParam"/>
<owl:disjointWith rdf:resource="#exclusionParam"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="follow">
<owl:inverseOf rdf:resource="#precede"/>
</owl:ObjectProperty>
```

```
<owl:Class rdf:ID="functions">
<rdfs:subClassOf rdf:resource="#linuxBckp_Operation"/>
<owl:disjointWith rdf:resource="#command"/>
<owl:disjointWith rdf:resource="#dataToBackup"/>
<owl:disjointWith rdf:resource="#options"/>
<owl:disjointWith rdf:resource="#storage"/>
</owl:Class>
<incremParam rdf:ID="g"/>
<owl:ObjectProperty rdf:ID="hasParam">
<owl:inverseOf rdf:resource="#isParamOf"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasPath">
<owl:inverseOf rdf:resource="#isPathOf"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasSetTime"/>
<owl:ObjectProperty rdf:ID="incompatibeWith">
<owl:inverseOf rdf:resource="#compatibleWith"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="incremParam">
<rdfs:subClassOf rdf:resource="#typeOfbackup"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="isParamOf">
<owl:inverseOf rdf:resource="#hasParam"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isPathOf">
<owl:inverseOf rdf:resource="#hasPath"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="keepOldFileParam">
<rdfs:subClassOf rdf:resource="#options"/>
<owl:disjointWith rdf:resource="#onlyStoreFileNewerthanParam"/>
<owl:disjointWith rdf:resource="#exclusionParam"/>
<owl:disjointWith rdf:resource="#preserveAccessParam"/>
<owl:disjointWith rdf:resource="#fileChangeLatestParam"/>
<owl:disjointWith rdf:resource="#archiveNameParam"/>
<owl:disjointWith rdf:resource="#compressionParam"/>
<owl:disjointWith rdf:resource="#permission"/>
<owl:disjointWith rdf:resource="#typeOfbackup"/>
<owl:disjointWith rdf:resource="#fileListParam"/>
<owl:disjointWith rdf:resource="#verificationParam"/>
<rdfs:comment rdf:datatype="&xsd;string"
>keep existing files; don&apos;t overwrite them from archive
</rdfs:comment>
</owl:Class>
<owl:Class rdf:ID="keepSameOwnerParam">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#incompatibeWith"/>
```

```
<owl:someValuesFrom rdf:resource="#noSameOwnerParam"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="#permission"/>
<owl:disjointWith rdf:resource="#preservePermissionParam"/>
<owl:disjointWith rdf:resource="#noSamePermissionParam"/>
<owl:disjointWith rdf:resource="#noSameOwnerParam"/>
</owl:Class>
<owl:Class rdf:ID="linuxBckp_Operation">
<owl:disjointWith rdf:resource="#time"/>
<owl:disjointWith rdf:resource="#exclusionFilePatern"/>
</owl:Class>
<fileChangeLatestParam rdf:ID="newer-mtime"/>
<noSameOwnerParam rdf:ID="no-same-owner"/>
<noSamePermissionParam rdf:ID="no-same-permission"/>
<owl:Class rdf:ID="noSameOwnerParam">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#incompatibeWith"/>
<owl:someValuesFrom rdf:resource="#keepSameOwnerParam"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="#permission"/>
<owl:disjointWith rdf:resource="#preservePermissionParam"/>
<owl:disjointWith rdf:resource="#noSamePermissionParam"/>
<owl:disjointWith rdf:resource="#keepSameOwnerParam"/>
</owl:Class>
<owl:Class rdf:ID="noSamePermissionParam">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#incompatibeWith"/>
<owl:someValuesFrom rdf:resource="#preservePermissionParam"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="#permission"/>
<owl:disjointWith rdf:resource="#noSameOwnerParam"/>
<owl:disjointWith rdf:resource="#keepSameOwnerParam"/>
<owl:disjointWith rdf:resource="#preservePermissionParam"/>
</owl:Class>
<owl:Class rdf:ID="onlyStoreFileNewerthanParam">
<rdfs:subClassOf rdf:resource="#options"/>
<owl:disjointWith rdf:resource="#verificationParam"/>
<owl:disjointWith rdf:resource="#typeOfbackup"/>
<owl:disjointWith rdf:resource="#archiveNameParam"/>
<owl:disjointWith rdf:resource="#keepOldFileParam"/>
<owl:disjointWith rdf:resource="#exclusionParam"/>
<owl:disjointWith rdf:resource="#fileChangeLatestParam"/>
```

```
<owl:disjointWith rdf:resource="#preserveAccessParam"/>
<owl:disjointWith rdf:resource="#compressionParam"/>
<owl:disjointWith rdf:resource="#fileListParam"/>
<owl:disjointWith rdf:resource="#permission"/>
<rdfs:comment rdf:datatype="&xsd;string"
>only store files newer than DATE</rdfs:comment>
</owl:Class>
<owl:Class rdf:ID="onlyStoreFileNewerThanTime">
<rdfs:subClassOf rdf:resource="#time"/>
<owl:disjointWith rdf:resource="#preserveAccessTime"/>
<owl:disjointWith rdf:resource="#fileChangeLatestTime"/>
</owl:Class>
<owl:Class rdf:ID="options">
<rdfs:subClassOf rdf:resource="#linuxBckp_Operation"/>
<owl:disjointWith rdf:resource="#command"/>
<owl:disjointWith rdf:resource="#dataToBackup"/>
<owl:disjointWith rdf:resource="#functions"/>
<owl:disjointWith rdf:resource="#storage"/>
</owl:Class>
<preservePermissionParam rdf:ID="p"/>
<owl:Class rdf:ID="permission">
<rdfs:subClassOf rdf:resource="#options"/>
<owl:disjointWith rdf:resource="#typeOfbackup"/>
<owl:disjointWith rdf:resource="#onlyStoreFileNewerthanParam"/>
<owl:disjointWith rdf:resource="#keepOldFileParam"/>
<owl:disjointWith rdf:resource="#archiveNameParam"/>
<owl:disjointWith rdf:resource="#fileChangeLatestParam"/>
<owl:disjointWith rdf:resource="#verificationParam"/>
<owl:disjointWith rdf:resource="#compressionParam"/>
<owl:disjointWith rdf:resource="#preserveAccessParam"/>
<owl:disjointWith rdf:resource="#fileListParam"/>
<owl:disjointWith rdf:resource="#exclusionParam"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="precede">
<owl:inverseOf rdf:resource="#follow"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="preserveAccessParam">
<rdfs:subClassOf rdf:resource="#options"/>
<owl:disjointWith rdf:resource="#exclusionParam"/>
<owl:disjointWith rdf:resource="#fileChangeLatestParam"/>
<owl:disjointWith rdf:resource="#keepOldFileParam"/>
<owl:disjointWith rdf:resource="#fileListParam"/>
<owl:disjointWith rdf:resource="#compressionParam"/>
<owl:disjointWith rdf:resource="#onlyStoreFileNewerthanParam"/>
<owl:disjointWith rdf:resource="#permission"/>
<owl:disjointWith rdf:resource="#typeOfbackup"/>
<owl:disjointWith rdf:resource="#verificationParam"/>
```

```
<owl:disjointWith rdf:resource="#archiveNameParam"/>
<rdfs:comment rdf:datatype="&xsd;string"
>don&apos;t change access times on dumped files</rdfs:comment>
</owl:Class>
<owl:Class rdf:ID="preserveAccessTime">
<rdfs:subClassOf rdf:resource="#time"/>
<owl:disjointWith rdf:resource="#onlyStoreFileNewerThanTime"/>
<owl:disjointWith rdf:resource="#fileChangeLatestTime"/>
</owl:Class>
<owl:Class rdf:ID="preservePermissionParam">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#incompatibeWith"/>
<owl:someValuesFrom rdf:resource="#noSamePermissionParam"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="#permission"/>
<owl:disjointWith rdf:resource="#noSameOwnerParam"/>
<owl:disjointWith rdf:resource="#keepSameOwnerParam"/>
<owl:disjointWith rdf:resource="#noSamePermissionParam"/>
</owl:Class>
<appendToEndOfStorage rdf:ID="r"/>
<keepSameOwnerParam rdf:ID="same-owner"/>
<owl:Class rdf:ID="singlefileName">
<rdfs:subClassOf rdf:resource="#dataToBackup"/>
<owl:disjointWith rdf:resource="#fileListName"/>
<owl:disjointWith rdf:resource="#directoryToBckpPath"/>
</owl:Class>
<owl:Class rdf:ID="storage">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#hasParam"/>
<owl:hasValue rdf:resource="#f"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="#linuxBckp_Operation"/>
<owl:disjointWith rdf:resource="#command"/>
<owl:disjointWith rdf:resource="#dataToBackup"/>
<owl:disjointWith rdf:resource="#functions"/>
<owl:disjointWith rdf:resource="#options"/>
</owl:Class>
<fileListParam rdf:ID="T"/>
<owl:Class rdf:ID="tapeFileName">
<rdfs:subClassOf rdf:resource="#storage"/>
<owl:disjointWith rdf:resource="#archivePath"/>
<owl:disjointWith rdf:resource="#archiveName"/>
</owl:Class>
```

```
<command rdf:ID="tar"/>
<owl:Class rdf:ID="time">
<owl:disjointWith rdf:resource="#linuxBckp_Operation"/>
<owl:disjointWith rdf:resource="#exclusionFilePatern"/>
</owl:Class>
<owl:Class rdf:ID="typeOfbackup">
<rdfs:subClassOf rdf:resource="#options"/>
<owl:disjointWith rdf:resource="#preserveAccessParam"/>
<owl:disjointWith rdf:resource="#archiveNameParam"/>
<owl:disjointWith rdf:resource="#fileChangeLatestParam"/>
<owl:disjointWith rdf:resource="#exclusionParam"/>
<owl:disjointWith rdf:resource="#keepOldFileParam"/>
<owl:disjointWith rdf:resource="#fileListParam"/>
<owl:disjointWith rdf:resource="#onlyStoreFileNewerthanParam"/>
<owl:disjointWith rdf:resource="#compressionParam"/>
<owl:disjointWith rdf:resource="#permission"/>
<owl:disjointWith rdf:resource="#verificationParam"/>
</owl:Class>
<updateParam rdf:ID="u"/>
<owl:Class rdf:ID="updateParam">
<rdfs:subClassOf rdf:resource="#functions"/>
<owl:disjointWith rdf:resource="#appendParam"/>
<owl:disjointWith rdf:resource="#appendToEndOfStorage"/>
<owl:disjointWith rdf:resource="#createArchiveParam"/>
<rdfs:comment rdf:datatype="&xsd;string"
>only append files that are newer than copy in archive
</rdfs:comment>
</owl:Class>
<owl:Class rdf:ID="verificationParam">
<rdfs:subClassOf rdf:resource="#options"/>
<owl:disjointWith rdf:resource="#keepOldFileParam"/>
<owl:disjointWith rdf:resource="#preserveAccessParam"/>
<owl:disjointWith rdf:resource="#permission"/>
<owl:disjointWith rdf:resource="#compressionParam"/>
<owl:disjointWith rdf:resource="#archiveNameParam"/>
<owl:disjointWith rdf:resource="#fileChangeLatestParam"/>
<owl:disjointWith rdf:resource="#typeOfbackup"/>
<owl:disjointWith rdf:resource="#exclusionParam"/>
<owl:disjointWith rdf:resource="#onlyStoreFileNewerthanParam"/>
<owl:disjointWith rdf:resource="#fileListParam"/>
<rdfs:comment rdf:datatype="&xsd;string"
>attempt to verify the archive after writing it</rdfs:comment>
</owl:Class>
<verificationParam rdf:ID="W"/>
<exclusionParam rdf:ID="X"/>
</rdf:RDF>
```