UNIVERSITY OF OSLO
Department of Informatics

Ontology for Host-based Anomaly Detection

Margareth P. Adaa
Oslo University College

May 23, 2007

Ontology for Host-based Anomaly Detection

Margareth P. Adaa
Oslo University College

May 23, 2007

Abstract

This project is about the description of ontologies for anomaly detection in computer systems. The special case of the anomaly detection system in Cfengine is used as a case study. Cfengine was designed at Oslo University College, based on a considerable body of research, and thus we have detailed insight into its operation. The Cfengine environment daemon collects many events in collaboration with cfagent that are presented to a system administrator for further analysis and countermeasures. In this work we want to make use of ontologies to structure the knowledge in a way that makes the process of reasoning about anomalies clearer. Ultimately, one could imagine that ontology capabilities would enable computers to perform automatic filtering process through inferencing and reasoning about their problem space.

Aknowledgement

I would like to express my genuine appreciation and special thanks to my Supervisor, Professor Mark Burgess, for his continued inspiration, dedication, and understanding throughout this work. I would also like to thank Professor Joan Serrat of Universitat Politcnica de Catalunya for his hospitality and support when I was in Barcelona for Ontology overview. Special thanks to Kyrre M. Begnum for helping to structure the direction of the project, valuable discussions and encouragement. I am very grateful for cooperation and support from fellow students, Matt Disney and Karim S. Ntieche. Thanks to all my friends, your support and encouragement is very much appreciated and would be always remembered. I would also like to thank the research team at RacerPro system for their generous release of the RacerPro reasoner software License for academic use. This work was conducted using the Protégé resource, which is supported by grant LM007885 from the United States National Library of Medicine. Special thanks and appreciation to my family: Dennis, Karen and Kevin, and my friends for your continued support and encouragement throughout the course up to now. This would not have been possible without you!

This work is supported by the EC IST-EMANICS Network of Excellence (#26854)

Contents

1 Introduction **1**
1.1 The concepts behind the problem 1
1.2 Motivation 3
1.3 Problem definition 3
1.4 Thesis outline 4

2 Background **6**
2.1 Some basic concepts 7
2.2 Knowledge Representation 8
2.3 Ontology 9
2.3.1 Features and roles of ontology 10
2.3.2 Categories of ontology 11
2.3.3 Reasoning in Ontology 12
2.3.4 Semantic Web 12
2.3.5 Ontology Representation Languages 13
2.3.6 Ontology modeling 14
2.4 Other Knowledge Representation formalisms 20
2.4.1 Topic Maps 20
2.4.2 UML 21
2.4.3 Promise Theory 22

3 What is anomaly detection? **24**
3.1 Network Monitoring and Observation 24
3.2 Anomaly Detection System 25
3.2.1 Introduction 25
3.2.2 State of the art 26
3.3 Cfengine anomaly detection 27
3.3.1 Host monitoring 28
3.3.2 Data Collection and Analysis 29
3.3.3 Tests 30
3.3.4 Observed variables and their accuracy 33
3.3.5 Towards variables classification 33
3.4 Other Anomaly Detection Systems 37

4 Ontology development **39**
4.1 Approach to behaviour discovery in events 39
4.1.1 Terminology identification 40
4.1.2 Formulating concepts relations 40
4.2 Implementation: Building the Ontology 41
4.2.1 Determining purpose, scope and requirements 42
4.2.2 Conceptual model 43
4.2.3 Semantic relationships 44
4.2.4 Relationships Identification 46
4.2.5 Implementing the ontology on Protégé 50
4.2.6 Some design decisions made 50

5 Results and discussion **56**
5.1 Evaluation process 56
5.2 Results 58
5.3 Alternative ontologies 59
5.3.1 A promise theory representation 60
5.3.2 Promise Theory approach to modeling Anomalies 62

6 Conclusions **69**
6.1 Future Work 70

A Sample event **72**

B Sample event log **74**

C List of Cfengine events **76**

D Graphical view of Results from Protege **86**

E RDF/XML **88**
E.0.1 110

List of Figures

2.1 Promise graph for an Observer . . . 23

3.1 Cfenvd base classes. . . . 29
3.2 Cfengine anomaly detection . . . 32
3.3 Cfengine variables . . . 34
3.4 Weekly Samba file sharing service . . . 35

4.1 Main concepts related to events . . . 40
4.2 Taxonomy of observations . . . 41
4.3 Sub-Ontology for variables . . . 42
4.4 Taxonomy of events . . . 43
4.5 Taxonomy of events . . . 45
4.6 Cfenvd base class source variables . . . 46
4.7 Individual, Class and Property representation . . . 52
4.8 Class definitions and descriptions. . . . 53
4.9 Excerpt showing class description in RDF/XML. . . . 53

5.1 Test Results from OWL Plugin. . . . 57
5.2 An example of a simple SPARQL query . . . 58
5.3 Results of Classifying the ontology . . . 59
5.4 Results of Run ontology tests . . . 60
5.5 Results of taxonomy classification . . . 61
5.6 Results of consistency check . . . 62
5.7 Inconsistencies . . . 63
5.8 Jambalaya view . . . 64
5.9 An example of a simple SPARQL query . . . 64
5.10 Consistency check of the ontology . . . 65
5.11 Classify taxonomy . . . 66
5.12 Classify taxonomy results . . . 67
5.13 Cfengine events class hierarchy. . . . 67
5.14 Promise graph for an observatory . . . 68

D.1 Consistency check . . . 86
D.2 Classify taxonomy . . . 87

List of Tables

3.1 Unix Tools 30

4.1 Additional constructors and their inverses 47
4.2 Example relationships 48
4.3 LDT relationships 49
4.4 Cfengine statistical classes 54
4.5 A sample of properties 55
4.6 Some synonym operators and their meaning 55

5.1 Some general conclusions about the events. 63

Chapter 1

Introduction

Anomaly detection is a subject that has been worked on for many years by many individuals. Possibly many tens (perhaps hundreds) of models have been proposed to define what is meant by an anomaly in a computer system. Some models are based on fault and reliability theory, others are based on the idea of intrusion or misuse detection. Many authors confuse the terms anomaly detection with Intrusion Detection. This makes it hard to understand what exactly is being discussed.

In spite of the numerous models, there is no standard approach to defining or detecting anomalies in computer systems, nor is there a particular system for anomaly detection that produces convincing results.

The aim of this project is to look at approaches toward mapping out this area of study, defining basic terminology and concepts and how they relate to each other. By looking at the concepts and their relationships we should be able to create a knowledge map for the field. We could then classify different works within these different concepts and terms and map one view of the problem into another.

This problem is too large however. In the time available for this project, only a small part of this can be accomplished. We therefore aim to create a framework that can be extended later, by looking at a single case study of a system well known at our college: Cfengine. The Cfengine anomaly detection system is based on a body of research [1, 2, 3, 4] so it gives us a clear opportunity for modeling anomalies completely.

1.1 The concepts behind the problem

Most researches and surveys done on anomaly detection are in the Intrusion Detection perceptive, meaning as one method for Intrusion Detection, the other common method being misuse or signature-based Intrusion Detection. Traditionally, anomaly detection is considered to be for detection of intrusions or

attacks. Most of existing researches and papers on anomaly detection look for anomalies in network behavior. However, recently the trend has moved towards host-based anomaly detection, and Cfengine is one of the systems using this approach.

The rationale behind the host-based approach is that, anomalous behaviour is of utmost concern regardless of its source (whether network traffic or locally at the host). From the security point of view, a host-based anomaly detection approach has the potential of detecting abnormal behaviour in a host that might indicate inside attacks as opposed to network-based anomaly detection which will not be able to detect such attacks because they do not generate network traffic. Additionally, what is considered normal in one host's environment could be different in another, hence a distinct model of "normal" behaviour need to be learned individually by each host.

Cfengine takes a broader perspective in anomaly detection by looking for abnormal behavior of a host which might include intrusions/attacks as well as non-malicious behavior. The information obtained from Cfengine anomaly detection system is intended for self-regulation of the system by initiating counter-response. For example, if the Cfengine anomaly detection system detects a sudden increase of the number of SMTP connections to be say three or more standard deviations above normal for a given time of week, this might indicate a possible spam attack and as a counter response, the decision of shutting down the mail server temporarily to avert the possible attack might be taken. Similarly, if an overuse of a certain service is detected, a decision of revoking that service temporarily - until the cause is known or the problem is solved - can be taken. However, the actions taken are as specified by policies.

The Cfengine anomaly detection is still in its infancy, and like other research and production anomaly detection systems, has some drawbacks. These include:

- too many events are produced by statistical analysis of collected data
- difficulty in identifying anomalous events

From the misuse (or signature based) Intrusion Detection perspective, the terms "false positive" and "false negative" can be defined as:

Definition 1 (False positive) *A false positive is when the system classifies an action as anomalous (a possible intrusion) when it is a legitimate action.*

Definition 2 (False negative) *A false negative is when an actual intrusive action has occurred but the system allows it to pass as non-intrusive behavior.*

However, we feel that, these terms can not be used in the same sense with anomaly detection because of the following reasons:

- Since anomaly detection is about detecting "abnormal (anomalous) behaviour", there is no clear and standard boundary or distinction between "normal" and "abnormal" behaviour. This leads to another problem,
- Difficulty in asserting whether a certain behaviour is anomalous or not, and being certain that is the case.

From this point on, we would be refering to "interesting" and "non-interesting" events rather than "false positive" or "false negative" as this is more appropriate for our case.

1.2 Motivation

What is an anomaly? We return to this question in more depth in chapter 3. In traditional anomaly detection systems where the focus is mainly in terms of security, a multitude of events are usually reported. In such systems, system administrators are overwhelmed by the multitude of events to be able to understand what the events are trying to tell. In such cases, the events are usually just stored for future reference, no analysis and correlation is done to understand what is really happening, hence intrusion attempts might go unnoticed and an attack might be successful. However, for the case of Cfengine anomaly detection, the events and alerts reported depends on the specified policies. There are too many events whose statistical values measured exceed the thresholds set by arbitrary policies. The challenge is to filter the "interesting" from the "non-interesting" events for further analysis.

The boundary between acceptable and anomalous behavior is much more difficult to define. This is because there is no distinct separation between normal and anomalous behavior. The most common way to draw this boundary is with statistical distributions having a mean and standard deviation. Once the distribution has been established, a boundary can be drawn using some number of standard deviations. If an observation lies at a point outside of the (parameterized) number of standard deviations, it is reported showing how much it deviate from the normal value in units of standard deviation. Cfengine employs this approach to detect anomalous behaviour but the technique is not optimal. There is still a need for further mining of information given by reported events to have a better understanding of the host's state.

1.3 Problem definition

The *cfenvd* is an environmental daemon in Cfengine which is used to collect statistical data about the recent history of each host (approximately the past two months), and classify them in such a way that they can be utilized by the

(cfagent). The data collected by the cfenvd are such as number of users; number of root processes; number of non-root processes; percentage disk full for root disk; number of incoming and outgoing sockets for netbiosns, netbiosdgm, netbiosssn, irc, Cfengine, nfsd, smt, www, ftp, ssh, and telnet.

Events have internal attributes having semantic interpretation, whose information once extracted, or inferred can be used to identify the meaning of an anomaly. The importance of classifying anomaly detection events has been emphasized by other researchers too. Kruger et al [12] wrote about Bayesian Event Classification for Intrusion Detection. Begnum et al [4] suggested that, one way of avoiding the multitude of “false positives” in anomaly detection is to use information content of events to classify events as interesting or not.

The problem we want to address with the present work is that of classifying the events collected from the ongoing project at Oslo University College, through the use of Ontology. We hope that, using the power of ontology by making use of computer-processable meaning (semantics), we can harness the power of Ontology in filtering interesting events from others. This will solve the problem of too many events reported. We also think that the Ontology might be able to provide more information about the host”s status.

More specifically, the present work will address the following problems:

- classify all events related to anomaly detection in Cfengine
- based on classification, define relationships between event variables and concepts
- use the relationships developed to filter, relate and infer information from events.

1.4 Thesis outline

The plan for the thesis is as follows.

- We begin by discussing the meaning of knowledge and knowledge representation as a basis for ontology, as well as few other knowledge representation methods.
- Next in chapter 3, we define the terminology of events and measurements, and define what we mean by an anomaly. Specifically, we explain about Cfengine Anomaly detection system, as a basis of our case study.
- In chapter 4, we describe the procedure of creating a conceptual model and present a small ontology using the OWL language via Protégé 2000 tool.

- In chapter 5 we presents and discuss results from ontology evaluation.
- Finally we discuss what is learned in this project and the main conclusions.

Chapter 2

Background

In this chapter we introduce and define the main concepts surrounding a subject of study - ontology.

Ontologies play an important role in information processing. As specifications of conceptualizations, they enable sharing terms across different applications and thereby provide a way for application cooperation. Ontologies are a basis for data sharing, data processing, and data integration. Ontological analysis clarifies the structure of knowledge. For a given a domain, its ontology forms the heart of any system of knowledge representation for that domain.

[20] introduce and advocate the use of ontologies for Information Security. In stating the case for using ontologies, they claim that an ontology organizes and systematizes all of the phenomena (intrusive behavior) at any level of detail, consequently reducing a large diversity of items to a smaller list of properties.

In [21]Undercoffer et al state the benefits of using ontologies instead of taxonomy, giving case scenarios within a distributed Intrusion Detection system. They also compare and contrast the IETF IDMEF (Intrusion Detection Message Exchange Format), an emerging standard that uses XML to define its data model, with a data model they constructed. Additionally, [22] in IDMEF argue that additional efforts are needed to provide a common ontology that lets all IDS sensors in a distributed environment to agree on what they observe.

The definitions included in the next section are based on the view of information and knowledge described in [5] and discussed in our research group[1]. Next, we briefly mention some of technologies compared or related to ontology.

[1]This section was developed from a discussion with Mark Burgess, Demissie Aredo, Thor Hasle and Karim Sani Ntieche

2.1 Some basic concepts

It is helpful to define some basic concepts. In particular we begin by defining information, knowledge, understanding and model. There are two reasons to make definitions like this (perhaps a little more formally than is necessary). One is to make a clean a separation of concepts and the other is to emphasize the important distinctions between concepts that seem similar but which actually have quite different meanings.

Definition 3 (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 4 (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.*

Knowledge is a very high level concept that includes human cognitive functions. Knowledge is associated with an individual or group of communicating individuals, because understanding and interpretation are *subjective*. The subjectivity of knowledge is one of the causes of *uncertainty* in communication.

All knowledge can be coded as information, so we can define knowledge simply as information which is coded. However, this avoids the important issue of interpretation and understanding.

Definition 5 (Understanding) *We define understanding to be the construction of a model that incorporates the elements of knowledge within a subjectively consistent framework.*

Since knowledge is, by this definition, assumed to be from that which has been understood, it must contain a model.

Definition 6 (Model) *A model is a collection of concepts, things (entities) and descriptions of their behaviours. It is any suitably idealized approximation to some phenomenon or system. A model is built on* assumptions *and leads to* consequences *or predictions.*

Since knowledge is subjective, different individuals can have different understandings or interpretations of the same set of facts, i.e. they have different models or world views.

2.2 Knowledge Representation

What are we trying to do with knowledge representation? This is an important subject in computer science, for programming (representing data in programs) and in management (analysing, reasoning and drawing conclusions about data). By introducing models we create a framework in which we can form hypotheses and either find support or disprove them. But we must be careful: computer science often muddles the concept of a *model* with that of an *architecture*. An architecture is a functional design. A model is an approximate representation of a system that makes a prediction about behaviour.

So for a modeling language we have the following requirements:

- The ability to organize information;
- The ability to reason about information;
- The ability to make predictions about behaviour

Below are some thoughts about three modeling frameworks. All can be used to describe architectures, but can they be used for understanding behaviour? Some basic questions:

- How do we represent knowledge?
- What is knowledge?
- Programming describes algorithms, not knowledge per se.
- Data-modeling describes stacked bundles of data, but not reasoning.
- Can we model behaviour? Is behaviour more than an algorithm?

In the field of Artificial Intelligence (AI), where ontology in computer science stems, Knowledge Representation (KR), aims at acquisition, modeling and storing of knowledge so that programs can process it. Most often, Knowledge Representation focuses either on the *representational formalism* or on the *information to be encoded* in it, also referred to as knowledge engineering. Ontology can be viewed as one method of Knowledge Representation.

An appropriate choice of a Knowledge Representation formalism, can simplify problem solving. This means, the choice of a particular type of KR formalisms depends on the type of domain knowledge.

Knowledge Representation techniques includes:

- Lists (e.g, linked lists that are used to represent hierarchical knowledge.)
- Trees (graphical method of representing hierarchical knowledge.)
- Rule-based representations (used in specific problem-solving contexts.)
- Logic-based representations (may use deductive or inductive reasoning.)

2.3 Ontology

Ontology is a term borrowed from Philosophy, which means the description of “how things are” (Greek “ontos” ($o\nu\tau o\zeta$ “being or existence” and “logos” $\lambda\alpha\gamma o\zeta$ “speech” or “meaning”). It is thus about describing the basic categories and relationships of being or existence for entities in a domain.

In other fields such as Knowledge Engineering, Software Engineering, and Artificial Intelligence, ontology has been defined differently by different communities and people. The most common quoted definition from the AI community is the one by Gruber,[13]:
”An ontology is an explicit specification of a conceptualization.” This meaning of ontology is used mostly in the context of knowledge sharing. Conceptualization is a key term in ontology and is defined as ”a set of objects which an observer thinks exist in the domain of interest and relations between them” [14]. To specify a conceptualization, concepts and relations are defined in terms of slots and axioms. Axioms are stated in order to constrain the possible interpretations of the defined terms to avoid ambiguities.

Since a body of formally represented knowledge is based on a *conceptualization*, one need to specify how the abstract conceptualization is represented as a concrete data structure, in order to manipulate this knowledge.

In the knowledge engineering perspective, ontology has been defined as ”a formal mechanism for specification of conceptualization into a shared domain”[18] This work will commit to the newer definition given by J. Strassner[17]:

```
An ontology is a formal, explicit specification of a shared, machine-re
vocabulary and meanings, in the form of various entities and relationsh
them, to describe knowledge about the contents of one or more related s
throughout the life cycle of its existence. These entities and relation
```

We must be careful to distinguish between an ontology and a representation of an ontology. To explain this, we must elaborate on what a representation is.

Definition 7 (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.*

Representation theory is a branch of mathematics that is concerned with finding and classifying all mappings or associations that satisfy the constraints of a given model (typically in group theory).

As mention previously, ontology can be viewed as knowledge representation formalism. Moreover, ontology employs some mechanisms to represent

it, such as Ontology representation languages like OWL, OIL(Ontology Interchange Language)[2] etc. See section 2.3.6 for more about ontology languages.

We can thus summarize by defining ontology as:

Definition 8 (Ontology) *Ontology is a declarative description of knowledge existing in a domain of interest, that is made sharable due to mutual understanding, through explication of implicit knowledge.*

2.3.1 Features and roles of ontology

One of the roles of an ontology is to provide vocabulary for metadata description with computer-understandable semantics. However, there are two large differences between the roles of an ontology for knowledge bases and those for metadata: One is philosophical and the other is practical. The philosophical one is that while an ontology for knowledge bases is a specification of the conceptualization of the target world, that for metadata is a set of computer-understandable vocabulary. The practical one is that an ontology for metadata does not have to consider the instance problem which is one of the most serious issues of an ontology for knowledge bases[15].

An Ontology can be used for different purposes, but when used for the purpose of enabling knowledge sharing and reuse, it is a specification used for making ontological commitments. Ontological commitments are agreements to use the shared vocabulary in a coherent and consistent manner[13]. Ontologies are designed such that agents commit to ontology to enable knowledge sharing among themselves. An agent is said to be committed to an ontology if its observable actions are consistent with the definitions in the ontology. However, an agent that commits to an ontology need not share a knowledge base and hence does not have to be able to answer all queries that can be formulated in the shared vocabulary, since each agent may know things others do not. In other words, a commitment to a common ontology is a guarantee of consistency, but not completeness, with respect to queries and assertions using the vocabulary defined in the ontology.

Ontology as a knowledge representation provides all necessary constructs that add semantics to information being represented. Ontologies are constructed using knowledge representation languages and logics, which enables agents to automatically make informed domain-dependent reasoning sing the knowledge captured by ontologies. Additionally, ontologies contain rules and axioms that help to define completely the values that a concept can have, which can be useful in describing certain behaviors.

Generally, an ontology consists of:

[2]OIL is a Web-based representation and inference layer for ontologies,

- **Concepts:** represent a conceptualization; the class of all the examples of that event or entity
- **Relations:** represent a relationship between concepts
- **Axioms:** express a necessary facts holding between concepts and relationships
- **Instances:** represent a specific Individual

Ontologies have many uses, including:

- allowing for more complete and accurate modeling of domain knowledge than data models, where assumptions can be explicitly defined;
- allowing readily reuse through equivalences and mappings;
- providing the means to describe knowledge in a form understandable to both humans and intelligent agents;
- can be used by rule-driven applications to make inferences from conceptual models;

2.3.2 Categories of ontology

Some researchers have categorized Ontologies as follows:

- Task Ontology - an ontology that formally specifies the terminology associated with the type of task, e.g. scheduling, planning etc.
- Method Ontology - ontology that formally specify the definitions of the relevant concepts and relations used for specifying the reasoning process (problem solving) to accomplish a task.
- Domain ontology - ontology defined for conceptualizing the particular domain, e.g. job-shop scheduling, nurse assignment, air-gate assignment etc.
- Application ontology - it contains the essential knowledge in order to model a particular application under consideration.

2.3.3 Reasoning in Ontology

Description logics (DLs) [28] are a family of knowledge representation languages that can be used to represent the knowledge of a domain of interest in a structured, formal and understandable way.

Description Logics based languages are commonly used to implement ontologies. Ontology as a knowledge representation formalism employing DL, represent knowledge of a particular domain by defining the relevant concepts of the domain (its terminology), and then use these concepts to specify properties of objects and individuals occurring in the domain (the world description).

Description Logics are known for their expressiveness and has clearly defined semantics. Description Logics capture the meaning of the data by concentrating on classes and properties and their relationships. An important characteristics of Description Logics worth mentioning is that of checking for inconsistencies and organization of the concepts on a taxonomy built automatically by a system, from the concept definitions.

Description Logics are first-order logic predicate calculus with ideas from semantic networks that allow hierarchical representation of classes and instantiations of terms and their relationships, called terminological box (**TBox**), and assertions over them, called assertional box (**ABox**)[19].

Description Logics reasoning mechanisms are based on *subsumption*, which determines whether a term is more general than another, and *instance recognition*, which determines all concepts and relations that an individual satisfies. Additionally, *completion mechanisms* which perform logical operations such as contradiction detection, incoherent term detection and inheritance, both for descriptions and assertions about individuals, completes the basic set of reasoning mechanisms provided by Description Logics systems.

The reasoning in this work is provided by the combination of the ontology language used, *OWL DL* which is a sub-language of OWL that is based in part on the description logic, and the reasoner RacerPro.

Reasoning is important in ontology because it is also used to ensure the quality of ontology. Through the use of a reasoner, it is possible to test whether concepts are non-contradictory and to derive implied relations, during ontology design.

2.3.4 Semantic Web

Semantic Web is an extension of the current World Wide Web whose web content contains documents with computer-processable meaning (semantics), such that software agents can understand, interpret, share and intergrate information more easily. In other words, the data in the Semantic Web is formally defined and linked to enable effective information discovery, integration, and reuse across various applications. Semantic Web uses descriptive technolo-

gies such as Resource Description Framework (RDF), RDF Schema (RDFS), Extensible Markup Language (XML) and Web Ontology Language (OWL), to classify data from multiple domains based on their properties and relations between them. This classification adds meaning to the web contents thus facilitating automated information gathering and searching by software agents. We can say that one goal of the semantic web is to facilitate the communication between machines, with the ultimate goal of making the web more useful for humans. The success of Semantic Web requires capture of "real world semantics", which is afforded by ontologies. The current choice for ontology representation is primarily Description Logics.

XML provides syntax to represent and describe information, creating structured documents. XML allows users to add structure to their documents using their own tags to annotate Web pages. However, XML lacks a semantic model since the meaning of the structure is not known. XML schema is a language for restricting the structure of XML documents. RDF is XML-based framework for representing information in the Web. RDF provides a means for adding semantics to a document. Information is in principle stored in RDF statements which are machine-understandable. RDF statements are also referred to as *triples*, and consists of: **subject** (corresponding to a resource); **predicate** (a property) and **object** (a property value). RDF Schema is an extensible knowledge representation language, for describing properties and classes of RDF resources (objects) with semantics. OWL is an enhanced RDF having more vocabulary for describing classes, properties, and relations between classes such as disjointness, equality, cardinality, symmetry, etc. OWL defines the *types* of relationships that can be expressed with RDF using XML vocabulary to indicate the hierarchies and relationships between different resources.

One goal of the semantic web is to facilitate the communication between machines and based on this, achieve another goal of making the web more useful for humans.

2.3.5 Ontology Representation Languages

In order for ontologies to be used within an application, they must be specified, in some formal representation so a allow shared understanding. The syntax for ontology language needs to be intuitive for human users and be compatible to existing standards (such as XML, RDF, and RDFS). In addition, the ontology language needs to have an expressive power that is just sufficient for defining the relevant concepts in enough detail, so that the reasoning ability is not affected.

A variety of languages exists that are used to represent conceptual models, each with varying expressiveness, ease of use and computational complexity. An example of these languages are SHOE, XOL, RDF, OIL and OWL.

Generally, these languages fall into three kinds namely,

- vocabularies defined using natural language;
- object-based knowledge representation languages such as frames and UML; and
- languages based on predicates expressed in logic such as Description Logics.

As mentioned previously, of these types of ontology languages, this work employs the last category. Specifically, we used Web Ontology Language (OWL), which is integrated with the ontology development tool used, Protégé 2000. Since the ontology representation language plays a vital part in design, use and capabilities of an ontology, we see it proper at this time to describe shortly, the Web Ontology language (OWL).

OWL has three sub-languages which differ according to the level of expressiveness, namely:

- OWL Lite - This is least expressive, used mainly for simple class hierarchy and constraints definition.
- OWL DL - This is more expressive than Lite. It is based on Description Logics hence can perform automated reasoning and compute the classification of hierarchies automatically as well as check for inconsistencies.
- OWL FULL - Used for situations requiring most expressiveness, even at the expense of guaranteed decidability (all computations will finish in finite time) and computational completeness (all conclusions are guaranteed to be computed), hence it is not possible to have complete reasoning for every feature of OWL Full.

We use OWL DL for this work, especially because we want to use its reasoning capabilities.

2.3.6 Ontology modeling

Some Definitions

An OWL ontology has the following components:

- **Individuals** - represents objects in the domain of interest. They are also referred to as instances of classes
- **Properties** - are binary relations between two individuals, linking them together.

- **Classes** - are sets containing individuals, described to precisely give the requirements for class membership.

OWL **Properties** are used to describe relations between two Individuals. There are three main type of the Properties:

- An *object* property linking an individual to another individual.
- A *datatype* property linking an individual to a data literal (e.g 32), having a *type* xml:integer.
- An *annotation* property, linking a class to a data literal (string).

OWL properties may have sub properties, so that it is possible to form hierarchies of properties (a subsumption hierarchy). Sub properties specialize their super properties in the same way that subclasses specialize their superclasses.

Note: *It is also possible to create sub-properties of datatype properties. However, it is not possible to mix and match object properties and datatype properties with regards to sub properties. For example, it is not possible to create an object property that is the sub-property of a datatype property and vice-versa.*

Class Description and Definition

The process of formulating class definitions that will constitute the ontology is one of the most central activities during ontology design. This is a nontrivial task since class definitions are specified using an expressive ontology language such as OWL, in a declarative fashion. Care must be taken during class definition because the ontology designed can easily be inconsistent where by there is no model that matches class definitions contained in the ontology (e.g a class that can not have any instances). For a example, an inconsistent ontology may result from an addition of a new class definition that does not interact with the existing ones as intended.

A class may contain a set of *Individuals* also referred to as *instances* of the class. An Individual can be a member of multiple classes because in OWL classes are assumed to overlap. In a particular case where two classes do not overlap, that is, there are no members belonging to both classes, it is important to specify this fact explicitly using the *disjoint* feature. Individuals are related to other objects and to data through *Properties*. A *property* is a way of describing a relationship that exists between Individuals and between Individual and data. We say, relationships are *formed along properties*. A model containing classes that has been made disjoint and structured in a hierarchy (subsumption) is still not semantically rich, but needs to be enriched through specification of relationships that exists between Individuals of different classes.

Class membership can be explicitly specified using two conditions, namely *necessary* and *necessary & sufficient*. *Necessary conditions* are conditions that must be fulfilled by Individuals to belong to that class. A set of necessary conditions is also referred to as a **Description**. *Necessary & Sufficient* conditions represent conditions that are not only necessary for class membership but also sufficient to determine that, any Individual (who is a member of any non-disjoint Class) that satisfies these *Necessary & Sufficient* conditions can be inferred to be a member of the class in question. Each set of necessary & sufficient conditions is an Equivalent Class, and all classes whose individuals satisfies these conditions are subclasses of the (inferred) Equivalent class. A sets of *Necessary and Sufficient* conditions is also referred to as **Definitions**.

Note: A class can have multiple sets of *Necessary and Sufficient* conditions(i.e multiple *definitions*).

Classes with *Necessary & sufficient* conditions are called **Defined classes** while those with only *Necessary* conditions are called **Primitive classes**. A *defined* class gives a *complete definition* of a particular class while the *primitive* class gives a *partial description* of a class.

Property Characteristics

In the process of describing relations existing between Individuals and between Individual and data using properties, we can use a number of property characteristics to add more semantics to properties. Protégé OWL allows specification of the following characteristics for properties:

Definition 9 (Functional Properties) *If a property is **functional**, for a given individual, there can be at most one individual that is related to another individual via the given property.*

If a functional property P relates Individual A to Individual B, then all relations along P relate Individual A to Individual B . Individual B could also be a datatype value. Functional properties are also known as *single valued properties* or *features*.

Definition 10 (Inverse Properties) *If a property P has its inverse, and P links Individual A to Individual B, then its inverse property will link Individual B to Individual A.*

Each object property may have a corresponding inverse property, and the inverse property will link the individual linked by the original property, in reverse direction.

Definition 11 (Inverse Functional Properties) *If a property is **inverse functional** then it means that the inverse property is functional also.*

Definition 12 (Transitive Properties) *If a property P is **transitive**, and the property relates Individual A to Individual B, and also Individual B to Individual C, then it can be inferred that Individual A is related to Individual C through the property P.*

Definition 13 (Symmetric Properties) *If a property P is **symmetric**, and the property relates Individual A to Individual B then Individual B is also related to Individual A through the same property P.*

We can see that, *symmetric* property is its own *inverse* property.

Property Restrictions

Properties are used to describe Individuals using *Restrictions*. Properties can be restricted in how they are used:
Globally - by describing or stating things about the property itself (e.g using Domain and Range);
Locally - by restricting their use for a particular class (i.e Class restrictions)

Property restrictions describes an anonymous class, which is a class of all individuals that satisfy the restriction. In OWL, there are two types of property restrictions, namely *value* constraints and *cardinality* constraints. For a particular class description, a *value* constraint restricts the range of property while, a cardinality constraints restricts the number of values a property can have. Property restrictions can be applied to both *object* property and *datatype* property. The use of Property restrictions is the primary way in which rules are written in Protégé . Protégé OWL has built-in OWL constructors as shown in Figure 4.6, of which $\forall$, $\exists$ and $\ni$ are used to specify value constraints, and they are local constraints. $\geq, \leq$ *and* $=$ are used to specify both *local* and *global* cardinality constraints.

***Note**: Global property constraints apply to all instances of the property, whereas local property constraints apply only to the class being described.*

Property Domain and Range

Whenever applicable, we specify *Domain* and *Range* for Properties not as constraints to be checked but rather as axioms for the Reasoner to use to make inferences. Errors in domain and range specification do not necessarily make ontology inconsistent or contain errors.

If a relation is:
subject-Individual ➔ hasProperty ➔ object-Individual
then, the *Domain* is the class of the subject-Individual and the *Range* is the class of the object-Individual (or a datatype if hasProperty is a Datatype Property).

Definition 14 (Domain) *The domain of a Property implies certain superclass-subclass relationships for classes that have that property. Any Individual (or class) that uses a property with a domain set can be inferred to be a member (or a subclass) of the domain class.*

Definition 15 (Range) *A range of a Property, implies certain superclass-subclass relationships for classes that share that particular property. Any Individual (or class) that uses a property with a range set can be inferred to be a member (or a subclass) of the range class.*

It is worthwhile to note that the understanding of domain and range in OWL and other Description Logic based Languages is somewhat different from that of programming languages or frame-based reasoning systems. In the later, the Domain and Range are used to verify the correctness of relationships by ensuring that a relation is only used in contexts that make sense. In the former, domain and range are used by the reasoner to infer additional information about classes and instances.

Tools

The tools used for developing and querying our ontology were Protégé 2000 version 3.2, a reasoner (RacerPro), and OWL Plugin. Protégé incorporates a number of plugins (e.g OWLviz, accessed through their respective tabs in the Protégé OWL editor, such as Ontoviz, Queries, OWLViz, TGviz etc. These tools provide different views and abilities to develop and manipulate an ontology being designed. Only a few of these tools were used with the present work due to time limitations.

Protégé

The Protégé is a free, Open Source ontology development and knowledge acquisition environment that provides users with tools to construct domain models and knowledge-based applications with ontologies. According to the authors [24], the Protégé platform supports two main ways of modeling ontologies namely, the Protégé -Frames and Protégé -OWL editors. Protégé ontologies can be exported into a variety of formats including RDF(S), OWL, and XML Schema. Additionally, Protégé is based on Java, is extensible, and provides a plug-and-play environment that makes it a flexible base for rapid prototyping and application development[24]. The architecture of Protégé consists of two main parts, a "model" part and a "view" part. The Protégé *model* is the internal representation mechanism for ontologies and knowledge bases, and the *view* components provide a user interface to display and manipulate the underlying model. The system was designed to be extensible through the use of plug-ins that allows enhancements to Protégé basic capabilities. This work utilized the Protégé -OWL editor. The Protégé -OWL editor is an extension of Protégé that supports the Web Ontology Language (OWL), which is the ontology language recommended by the World Wide Web Consortium (W3C) to promote the development of the Semantic Web.

Among other things, the Protégé -OWL editor allows one to:

- Load and save OWL and RDF ontologies.

- Edit and visualize classes and properties.
- Execute reasoners such as description logic Reasoners.

Additionaly, Protégé -OWL supports the use of reasoners implementing the DIG interface and can only connect to reasoners over an http: connection.

OWL Plugin

The OWL plugin is an extension of the Protégé which is used to edit OWL ontologies, to access description logic (DL) reasoners, and to acquire instances for knowledge base creation.

RacerPro

RacerPro stands for Renamed ABox and Concept Expression Reasoner Professional. This is one of description logic reasoners that was used with this work, others are such as Pellet[3], FACT++[4] and KAON2[5]. RacerPro is a commercial tool with different types of licences one of which is a free semester license (180 days) for educational and research purposes. The author of the present work obtained such a licence for this work. RacerPro is available as an executable server for Linux, Windows, and MacOS X whereas the Windows version was used for the this work. It is worthwhile to note that, the standard RacerPro has almost no user interface, it just prints some welcome messages and basic status reports into the console or terminal window. Usually, all interactions with RacerPro can be done through network protocols like HTTP (DIG) or Racer native commands (over TCP/IP). RacerPro has a graphical user interface called RacerPorter for connecting and managing RacerPro servers. We used RacerPorter to access and use the reasoner. Detailed description of RacerPro is outside the scope of this work, however, we would like to mention some of the features of RacerPro version 1.9 that were utilized in relation with our work, which includes:

- Checking the consistency of an OWL ontology and a set of data descriptions.
- Finding implicit subclass relationships induced by the declaration in the ontology.
- Computing inferred hierarchy of the ontology.

Specifically, we used RacerPro to statically check our ontology for inconsistencies and for computing inferred hierarchy to check for the suggested changes. With RacerPro, one could also submit queries in order to verify their validity. The queries need to be expressed in the new Racer Query Language (nRQL), which is a description logic query language for retrieving individuals from an A-box (a set of assertions about Individuals) according to specific conditions. The communication between Protégé and

[3]See http://pellet.owldl.com/
[4]See http://owl.man.ac.uk/factplusplus/
[5]See http://kaon2.semanticweb.org/

Racer is done through the RQL Tab plug-in, which allows the OWL plug-in to send queries to Racer and receive the answers (results). A description of nRQL's syntax is beyond the scope of this work, but the interested reader is referred to [**?**]. Due to the rather steep learning curve of nRQL syntax and the time constraints, we were not able to test the ontology developed through queries.

OWLViz

OWLViz is designed to be used with the Protégé OWL plugin to enable the class hierarchies in an OWL Ontology to be viewed and incrementally navigated, allowing comparison of the asserted class hierarchy and the inferred class hierarchy. OWLViz requires Graph visualization (Graphviz), which is an open source graph visualization software, used to represent structural information as diagrams of abstract graphs and networks.

2.4 Other Knowledge Representation formalisms

The use of ontologies in Computer science and Software engineering fields is relatively new, so there has been some speculations as to the need for ontologies. It is not possible to list all technologies that are compared/contrasted to ontology, but this section will describe some of the common ones.

Some of existing Knowledge Representation (KR) formalisms such as Information models(e.g, Topic Maps) and Conceptual modeling languages (e.g UML) are often compared to ontology. The reason for this might be because they have some common features. For example, a *Relational Database Schema* defines a set of terms using **classes** (corresponding to *tables*, where terms are represented as the rows in a table), **properties** (attributes) (specified as *columns* in the table), and a limited set of **relations** between classes (corresponding to *foreign keys*).

An *object-oriented software model* defines a set of concepts and terms through a hierarchy of *classes* and *attributes* and a broad set of *binary relations* among those classes. Some *constraints* and other behavioral characteristics may be specified through methods on the classes or objects. A knowledge-representation system such as Ontology has the ability to express in addition, n-ary relations, rules, restrictions on classes and logical operations such as negation and disjunction. Another form of knowledge representation we looked into briefly was Promise Theory.

2.4.1 Topic Maps

Topic maps are an ISO standard for the representation and interchange of knowledge. Topic maps describes knowledge structures and associate them with information resources to make the information in them findable. Topic maps represents information using:

- *topics*: These are objects of interest;
- *associations*: These are relationships between them;

- *occurences*: Relationships between topics and relevant information resources.

Topic maps are similar to *concept maps*, *mind maps* and *semantic networks* in the sense that they all represent knowledge in their different forms.

An excellent summarized description of Topic Maps as a model of Knowledge Representation is given by Kazienko et al [29] and is quoted here. A Topic Map is based on : issues:

- extraction of *topics* (subjects) which are concepts typical for modeling a domain of knowledge,
- defining *associations* (relations) among topics,
- linking topics with a data layer (resources).

Each topic can have a *name* (none, one or more) and should have one or more *topic types*. A relation between topics and topic types is a simple class-instance association. Links between topics and their related information (e.g. web resources) are defined by objects called *occurrences*. The linked resource can be located in or outside the map. Occurrences like topics can be of a certain *type*. Types of occurrences are also defined as topics. There is a possibility to define relations between topics which are called *associations*. Each association can have an *association type* which is also a topic. There is no constraint about how many topics can be related by one association. Topics can play specific roles in association, described by *association role types* which are also topics. *Scopes* are assigned to topics, occurrences or associations, when one needs to define constraints to explain when they are valid. Topic maps provide also a mechanism which allows identifying seemingly disparate topics. Each topic can have a unique subject identity which describe topic in an unambiguous way. Subject identity is used for topic map merging when there is a need to recognize which topics describe the same subject.

2.4.2 UML

The Unified Modeling Language (UML) was created to be a specification language for programming,i.e a way of representing requirements and tests in an abstract form. UML provides a collection of modeling constructs and an associated graphical notation that can be used for modeling software, as well as for modeling the problem domain that of a system. However, since UML was developed based mainly from implementation perspective, it lacks the theoretical foundations for modeling real world domain. UML is criticized to have limitations such as ambiguity, inconsistency, inadequacy, and complexity in relation to conceptual modeling. These limitations are thought by some people to be due to the implementation-oriented design of UML whose constructs makes it inadequate for conceptual modeling of real-world domains. The most common criticism is that UML modeling does not

UML models are often used to specify software products and typically each product has its own model. UML models use graphical notation (diagrams) and UML 2.0 has 13 types of such diagrams. Some of the most useful, standard UML diagrams

are includes: use case diagram, class diagram, sequence diagram, statechart diagram, activity diagram, component diagram, and deployment diagram.

UML is a standard from the Object Management Group (OMG) and has a very large and rapidly expanding user community in the field of software engineering. In recent years there has been increasing efforts to bring together the Semantic Web technologies (such as RDF and OWL) and Software Engineering methodologies and languages[30]. An example of this kind of effort is shown by the OMG's Ontology Definition Metamodel (ODM). In spite of criticism against UML there has been a growing interest among some researchers about using UML as a Knowledge Representation language, specifically to represent ontology.

2.4.3 Promise Theory

Promise theory is a high level description of "agent" behaviour. Agents in promise theory are truly autonomous entities: they are entities who decide their own behaviour, cannot be forced into behaviour externally but can voluntarily cooperate with one another[**?**]. A promise is a directed edge $a_1 \xrightarrow{b} a_2$ that consists of a promiser a_1 (sender), a promisee a_2 (recipient) and a promise body b, which describes the nature of the promise. Promises made by agents fall into two basic categories, promises to provide something or offer a behaviour b (written $a_1 \xrightarrow{+b} a_2$), and promises to accept something or make use of another's promise of behaviour b (written $a_2 \xrightarrow{-b} a_1$). A successful transfer of the promised exchange involves both of these promises, as an agent can freely decline to be informed of the other's behaviour or receive the service.

Promises can be made about any subject that relates to the behaviour of the promising agent, but agents cannot make promises about each others' behaviours. The subject of a promise is represented by the promise body b.

The essential assumption of promise theory is that all nodes are independent agents, with only private knowledge (e.g. of time). No node can be forced to promise anything or behave in any way by an outside agent. Moreover, there are no common standards of knowledge (such as knowing the time of day) without explicit promises being made to yield this information from a source. What makes promise theory interesting for ontology is that promises themselves have to be organized into an ontology of types, but here the types are motivated very pragmatically by what one promises will happen in the system. Unlike languages designed for ontology development, many details about attributes are omitted in promise theory, assumed to be "inside" the agents (out of sight). Instead the focus is on what are the necessary and sufficient promises to predict certain behaviour. Since anomaly detection is also about verifying behaviour it is not unnatural to expect promise theory to have a useful viewpoint on the problem.

Looking at aspects of ontology and Promise Theory we found that the two has some interesting similarities and differences, namely:

- Where as Ontology focus on all knowledge inherent in concepts according to the scope of the domain of interest, Promises focus mainly on the agreements made between agents. Any other private knowledge inherent in an agent that

is not part of the agreement is ignored. This can be seen as partial knowledge, more suitable for modeling Task ontologies[6] or SOA[7] applications.

- Ontology like Promises has *directional* relations. It must be noted however that, some relations in ontology has *inverses* whereby each direction of a relation is a separate relation. (e.g hasPart and isPartOf are t wo separate relations). In addition, some relations are symmetrical, which might be confused as bidirectional but each direction is depicted by the domain and range of that relation.

- Another noted similarity between Promises and ontology is that they both focus on *instances* while describing relations. It should be noted that, sometimes it is said that relations in ontology are described between classes. This is valid because actually, classes are a set of Individuals and by saying classes are related to other classes it means Individuals(instances) of one class are related to Individuals of another class (through a relation).

An example of a Promise graph with relations is shown in Figure 2.1. From the

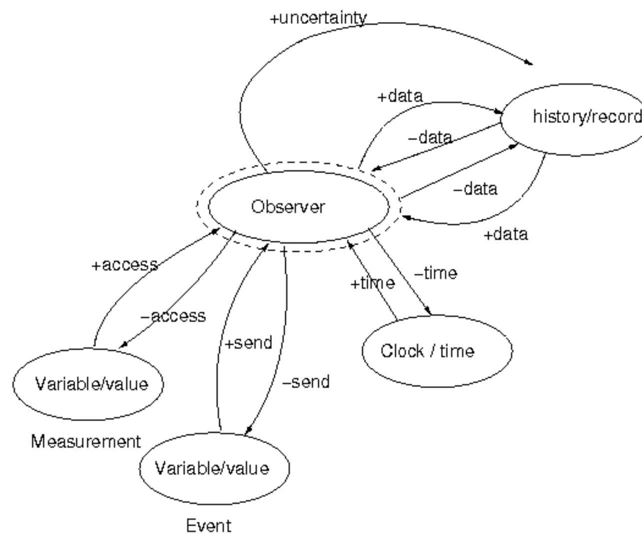

Figure 2.1: Promise graph for an Observer

Promise graph we can see that, by following the chains of dependencies, one can see the functional processes that relates agents. This can be a form of reasoning knowledge which can be used in ontology. We believe that further research on ontology and Promise Theory might reveal complimentary features suitable for knowledge representation in Anomaly Detection as well as in other domains.

[6]Task Ontology is an ontology that formally specifies the terminology associated with the type of task, e.g. scheduling, providing services, planning etc.

[7]Service Oriented Architecture

Chapter 3

What is anomaly detection?

What is an anomaly? In this chapter we consider how to define this concept as this obvious step is often taken for granted. On one hand it is easy to find unexpected behaviour in a system, if the threshold for surprised is low. On the other hand, If our threshold for surprise is higher, we expect fewer cases. It is therefore clear that there must be some subjective choice involved in defining an anomaly.

We begin by defining some terminology.

Definition 16 (Event) *An event is an occurrence of a data value from some measuring instrument at a time that is not determined by the observer. It is an unprogrammed data point.*

More specifically in relation to this work, we refer to the term *event* as a logged occurrence from a sensor, such as cfenvd.

Definition 17 (Measurement) *A measurement is a purposeful act to acquire the value from some measuring instrument. Data are collected at a time determined by the observer.*

Anomaly means "without name" (unknown), i.e. it is literally something that we cannot classify. Its meaning in Computer Science is less literal, since one of the things we want to achieve is a classification of different types of anomalies.

A more realistic definition is:

Definition 18 (Anomaly) *An observation that does not fall within specified constraints (i.e. policy).*

3.1 Network Monitoring and Observation

In computer networks, monitoring (host or network) is done for a variety of reasons, such as performance checking, determination of resource usage status in grid environments, etc.

Network monitoring refers to systems that simply observe and report on a network, without taking any corrective action of their own accord. Network Monitoring when used in conjunction with Anomaly Detection tools, has the potential of giving

alerts about security breaches and intrusions by detecting sudden changes in usage pattern and traffic behavior. In recent times, network monitoring and Intrusion Detection have become an integral part of a network security. For example, Host based Intrusion Detection Systems typically monitors system, event, and security logs on Windows environment and syslog in Unix environment. Network based Intrusion Detection System monitors all traffic in real time as it travels across the network, and analysis of the data can be done online or offline.

There exists a number of Industry monitoring platforms and softwares, both proprietary and open source such as Nagios, Zenoss etc. Zenoss is a network and systems monitoring platform that is Python-based and is a free, open source download for Linux. Nagios is an open source host, network, and service-monitoring system, that monitors network services (SMTP, POP3, HTTP, NNTP, PING, etc.); host resources (processor load, disk and memory usage, running processes, log files, etc.) and can even monitor environmental factors such as temperature.

Making resource profiles involves monitoring of system wide usage of resources such as applications, accounts, communications ports, protocols, storage media,etc. This is a necessary step in developing historic usage profile that can be used to detect variations from the normal profile. Detection of anomalous system and programs behaviour involves defining of the variables to be monitored and defining the criteria for anomalous behavior, for each variable. The criteria may take different forms such as ranges of values considered to be out of "normal" values. Anomaly detection with Cfengine uses a statistical model that detects and classifies the measured number of events in units of standard deviation.

Typical variables used for detecting anomalous system behavior are those associated with performance monitoring. Cfengine uses variables as shown in [**?**], but alternatively variables such as time spent in certain program functions, patterns of memory usage, quantity and destinations of network communications, and time spent in code representing specific operating system services can be used.

3.2 Anomaly Detection System

3.2.1 Introduction

Intrusion Detection Systems employ network and system monitoring softwares and hardwares to analyse streams of monitored data. Generally, IDS can be defined as a security system consisting of tools, methods and resources that monitors system and network traffic in order to identify, analyse and report possible attacks from both inside and outside an organization. Intrusion Detection Systems can also be categorized as network-based, which deals with network traffic; and host-based, where operating system events are monitored. The term "Intrusion Detection" is often used to encompass both anomaly detection (deviation from normal behaviour) and misuse detection (detection of known types of misuse), but in most cases "Intrusion Detection" suggests only the *detection of intrusions*. The common detection modes/ techniques are:

- misuse detection

- anomaly detection
- specification-based detection

misuse detection (also referred to as signature-based detection) uses the stored signature of known attacks to compare with the observed behaviour of current data, giving an alert if a match is found.

anomaly detection technique uses a pre-defined notion/standard of normal behaviour (profile) for comparison with monitored data, significant deviations from this baseline or threshold of normal behaviour is considered anomalous. Usually the stored profiles are constantly being updated in order to reflect changes in user or system behavior.

specification-based detection uses manually developed specifications to characterize legitimate program behaviours and deviations from legitimate behaviours are flagged as anomalous.

As agreed by many researchers in this field, the main advantage of misuse detection is that it can accurately detect known attacks, but has the disadvantage of failing to detect new attacks. Anomaly detection overcomes the limitation of misuse detection by focusing on "normal" system behaviours, which means it can detect new attacks. Its main drawback is the high rate of false alerts it produces, since previous unseen and yet legitimate system behaviours are also flagged as anomalous. Specification-based detection has the potential of detecting novel attacks, but unlike anomaly detection, its false positive rate can be comparable to that of misuse detection since it does not generate false alerts when unusual but legitimate program behaviour is encountered.

In most Anomaly Detection Systems, the behaviour of a system element (e.g. a user, a program, or a network element, etc.) is observed through the available audit data logs. The basic assumption in Anomaly Detection is that there is an intrinsic pattern or regularity in audit data that is consistent with the normal behavior which is different from the abnormal behaviour. It is important also that the system is able to adapt to changes in system and user behaviour over time.

Anomaly Detection may use one of these analysis procedures:

- quantitative analysis
- statistical measurement
- rule-based systems
- neural networks

3.2.2 State of the art

Most of the existing Anomaly Detection Systems are focused mostly on the security aspect, protecting a host and /or network from exploits and misuse. In most research and papers written, Anomaly Detection is categorized as one of the two most common approaches to Intrusion Detection, the other being misuse (Signature-based)detection.

Currently, Anomaly Detection is often incorporated to some extent in available Intrusion Detection Systems or products.

The main problem with anomaly detection systems is that it can detect an anomalous event but can not describe what it is. Most often, especially with Statistical Anomaly Detection techniques, a multitude of events requires the attention of the system administrator for analysis, a task which is nontrivial.

Since it is difficult to manually identify discrete events indicating anomalous behaviour in systems, several approaches to anomaly detection have been considered, such as

- Machine learning techniques
- Data mining
- Clustering
- Ontology-based analysis

3.3 Cfengine anomaly detection

Cfengine's scope of anomaly detection is not restricted to *Intrusion Detection* sense only. Cfengine is more concerned with detection of resource anomalies such as CPU, disk usage, memory, number of users, etc. The advantages of this approach in addition to detecting anomalous events - which might be intrusive or attacks, is the ability to tune and configure the system better, and be able to establish a baseline of "normal" behaviour. This section will try to describe our context of anomaly detection, specifically related to Cfengine. The two techniques (also referred to as *Tests*) used with Cfengine anomaly detection, (Two-dimensional time-series and Leap Detection Test) will be described.

With Cfengine, anomaly detection involves monitoring the host resources and services in order to detect any variation from the norm. The idea here is not only to look for anomalous behavior that can be indicative of intrusion or a compromised system, but also for behaviors that can give important information about problems existing within the system such as bottlenecks in resources, scheduling of certain activities like backups, etc. System behavior in this context includes things such as: uses of system resources like CPU load, disk space, etc.; usage patterns of services and protocols, user behavior or a combination of system variables.

In version 2.x and up of Cfengine, an environment daemon, cfenvd can be run in each host to measure system resource usage, independently of the other parts and records it in a database. This data then becomes the profile ('normal' behaviour) of the host. A training period of about two months is required to build up enough data for stable characterization of a host's behaviour.

The daemon collects the following long-term data: number of users, number of root processes, number of non-root processes, percentage disk full for root disk, number of incoming and outgoing sockets for netbiosns, netbiosdgm, netbiosssn, irc, Cfengine,

nfsd, smtp, www, ftp, ssh and telnet. These data can come from netstat, tcpdump, ps etc running on a host.

The collection of data by cfenvd is done approximately every 2.5 minutes, and the output is imported into cfagent. Using the specified rules in the policy file, cfagent creates corresponding classes to give events every 30 minutes and may give alerts or perform a predefined action. However, the cfagent only reports events whose classes are specified in the policy file without showing the frequency of occurrence of events.

3.3.1 Host monitoring

Cfengine consists of several components which together performs system administration tasks, namely:

- cfagent - An autonomous configuration agent,
- cfservd - A file server and remote activation service,
- cfexecd - A scheduling and report service,
- cfenvd - An anomaly detection service,
- cfenvgraph - Ancillary tool for cfenvd,
- cfkey - Key generation tool.

When *cfenvd* is run, it performs basic resource monitoring and acts as an anomaly detection engine. Cfenvd updates its measurements every two and a half minutes and has the ability to:

- learn behaviour trends in each host over a period of time(e.g four weeks)
- evaluate the current state of resources as compared to learned averages
- classify the current state of resources as compared to learned averages into classes to be used by cfagent

Figure 3.1 shows cfenvd base classes and suffixes where "base class" in the drawing represents each of the following variables: users; rootprocs; otherprocs; diskfree; incoming and outgoing: netbiosns, netbiosdgm, netbiosssn, irc, Cfengine, nfsd, smt, www, ftp, ssh, wwws, icmp, udp, dns, tcpsyn, tcpack, tcpfin, and tcpmisc.

The cfenvd stores the data in */var/Cfengine/state* and cfagent reads the current state of resources from a *env_data*, in the same directory. For each variable mentioned previously, the cfenvd stores the following sets of data:

- variable.***q*** - the latest raw value *q* measured, with format: *x y*
- variabe.***E-sigma*** - the computed average value with standard deviation, with format:*x y dy*
- variable.***distr*** - the distribution about the mean, a frequency histogram with format: *x y*

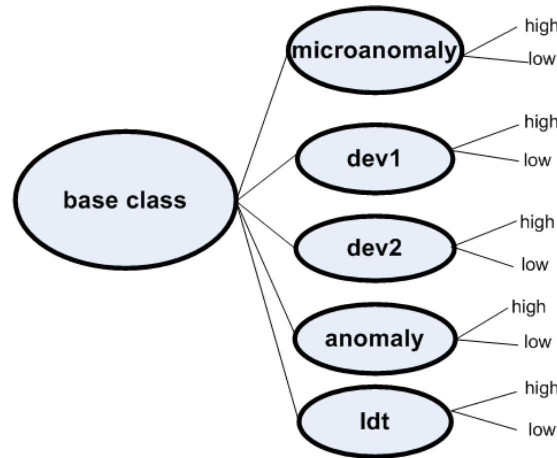

Figure 3.1: Cfenvd base classes.

The alerts about anomalies depends on policy decisions as specified in the cfagent configuration file. The measured data is stored in database but can be viewed in gnuplot after running the command **cfenvgraph -s**, which generates a directory of files that show a weekly snapshot of the system.

As an example of system resources monitoring, a *File System* behaviour can be obtained through the use of the *cfagent* which performs *file system* scan occasionally as per policy, and the results can similarly be viewed using *cfenvgraph and gnuplot*.
Note: *File system scan is resource intensive hence should not be done on a continuous basis.*

Simply collecting data and viewing it using cfenvgraph is not enough to understand the current state of the system. We need to know how our system works in order to understand the patterns shown by *cgenvgraph*, such as how the resources are being used by processes and users. However, full interpretation of data collected is still a problem, and this work is about trying to interpret the collected data in the form of events through the use of ontology.

3.3.2 Data Collection and Analysis

The present work is an extension to the project in Anomaly Detection with Cfengine done Oslo University College, specifically working on events collected. This section presents to the reader the data collection approach and variable selection mechanisms.

Cfengine employs statistical analysis in the detection of anomalies, thus in defining the current state of a host, a set of variables that were thought to be more straightforward, having a theoretical model with few unknowns were chosen. Specifically, variables chosen for characterization of system state were those having long-term periodicity that closely follow human patterns of behavior as well as those whose trend is affected by environmental parameters.

Collection of data was done in a similar way as the measurements done during system performance tuning, only that it was for the purpose of monitoring and characterizing a host state. Bearing in mind that, the process of data collection itself may

Table 3.1: Unix Tools

Tool	Information
ps	Process numbers, user numbers
vmstat	Memory and disk info, paging rates, etc.
iostat	Disk I/O specifics
netstat	Socket numbers
df	Disk space free on various partitions

have significant effect on system performance, a small daemon in C was written and used to collect data in the background in order to minimize the overhead. In most cases, the interesting data lies in the System kernel, and since there is neither standard for kernel variables, nor tools that access their values with standardized output, the experiments were restricted to a single OS (UNIX based). Table 3.1 shows the tools used to collect data.

A host state can be characterized at different level or resolution depending on the purpose for which characterization is meant for. Measured data can be represented in either discrete or continuous format depending on how the data is subsequently analyzed. Cfengine utilizes the continuous representation of data in the form of continuous curves that take into account *uncertainties* in measurements depicted by error bars, the measured values quoted in units of standard deviation, for long-term characterization of system state.

3.3.3 Tests

The following are two approaches used in relation with Anomaly detection project at Oslo University College using Cfengine. The present work is related to these two approaches as will be described in the following sections.

Two-dimensional Time-series Technique

This approach is used to detect anomalous behaviour on a single host through an iterative algorithm which has been encoded into Cfengine"s environmental daemon, cfenvd. Since time series can be expensive in terms of CPU time and disk space for data storage, the iterative algorithm and the use of random access database has been shown to achieve approximately tenfold compression of data as well as several orders of magnitude of CPU computation time can be spared [3]. The cfenvd continually updates the database of system average and variances, thus characterizing the "normal" behaviour. This database has the size of 2MB. The collected data is classified by cfenvd into classes, and is used to compare the incoming stream of data with the corresponding stored data to determine the current state of the host in relation to its recent history. The classes describe whether a parameter is above or below its average value, and how far from the average the current value is, in units of the standard-deviation

[11]. Cfagent then receives the classified data as a ‘class event’ which depending on specified policies can be used to determine countermeasures or follow-up responses for the state concerned. Specifically, cfenvd classifies the current state into the following levels: ***high*** or ***low***

- **dev1** means that the current level is at least one standard deviation above average.
- **dev2** means that the current level is at least two standard deviations above average.
- **anomaly** means that the current level is more than 3 standard deviations above average.
- **microanomaly** is used to describe values that are 2 standard deviations above normal, when the delta of the change is less than an arbitrary value of 5.

Note: *"normal" means that the current level is less than one standard deviation above average.*

For example, cfenvd sets classes such as:

```
RootProcs_high_dev2
www_in_high_dev2
netbiosssn_in_high_dev2
smtp_in_high_dev2
nfsd_in_high_dev2
```

every 2.5 minutes, which are imported to the cfagent. The cfagent runs approximately every 30 minutes and sets corresponding classes of events, one per each imported class. The order of arrival of events is not considered. It is worth noting that only events that are specified in the policies are collected by the cfagent. Additionally, cfagent doesn’t show the frequency of occurrence of an event. This information could be very useful in providing more understanding of the anomalous behaviour encountered, and in the decision of the proper action to be taken. Through the use of two-dimensional time-series analysis, Cfengine provides long-term anomaly detection, for the maintenance of host state and its adaptability to changing demands.

Realizing the importance of providing extensive information about anomaly characteristics, and in addition to specifying the statistical number of anomalies, Cfengine provides a second level of events filtering by employing entropy. In particular, Cfengine provides a measure of the entropy of the source IP addresses for the measured data. According to [11], a *low entropy* value means that the events came from only a few (or one) IP addresses. A *high entropy* value implies that the events are spread over many IP sources. These conditions are described by classes in the form:

```
entropy_smtp_in_low
entropy_www_in_high
```

This added information is quite helpful in describing events characteristics and provides an added level of understanding of events. For example, the

```
entropy_smtp_in_low
```

class will be set if smtp traffic is coming from one or two IP addresses, which will more likely be a spam or some kind of an attack. The

```
entropy_www_in_high
```

class will be set if www incoming traffic at the peak event of last data sample was spread evenly over all the incoming IP addresses. This event indicates that the resource usage was not from a single source, for example an attacker in a single location, but being evenly spread may be just a coincidental occurrence. What we see here is that, the added information that can be used to identify what is meant by an anomaly, so as to have a proper course of action taken. However, it is not possible to ascertain with certainty, whether the event is anomalous or not. This is what we referred to when pointing out the issue of "false positive" terminology being inappropriate in this case.

Figure 3.2 shows how cfenvd and cfagent work together to detect anomalies:

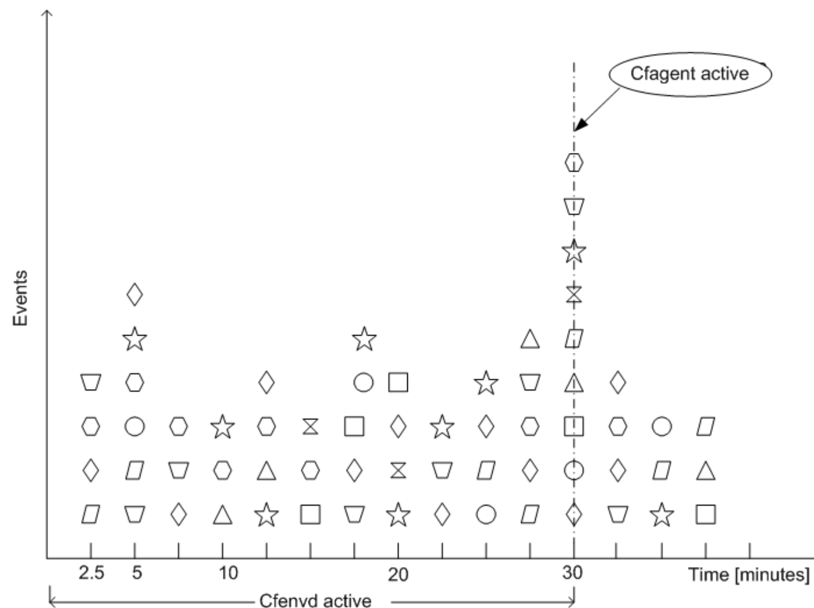

Figure 3.2: Cfengine anomaly detection

Leap Detection Test (LDT)

Whereas in Two-dimensional time-series test, the profile is generated in weekly basis, and deviation at any particular time is compared to a corresponding time from

previous weeks, the Leap Detection test methodology assumes neither periodic nor long-term behaviour.

The LDT test uses Chi-squared test to compare the random distributed data collected by the cfenvd. The hypothesis addressed is that: *the latest value is a significant leap of the observed population so far*. The leap-detection test for time series data uses the following formula:

$$\chi^2 = \frac{(x_1 + x_2 + \ldots + x_i - i * x_{i+1})^2}{i * (i+1) * \bar{x}}$$

where $x_1, x_2, ..., x_i$ are previously observed values in the time series and x_{i+1} is the most recent observation. The value of i therefore denotes the size of the memory. The mean as denoted by x includes all i + 1 values.

Depending on the wanted confidence, a trigger value is set which is compared to the chi-squared value obtained. Any value for chi-square test above this would cause the hypothesis to be true. This means for anomaly detection, every new observed value is tested against the hypothesis, and if the obtained chi-squared value is greater than the trigger value, an anomaly is detected.

The LDT test is incorporated into cfenvd and uses a sliding window with duration of 25 minutes, and only the data within this time frame is used in the computation. From the formula we see that the order and position of observed data has no effect on the computation and determination of an anomaly. This means that this test is more suitable for stochastic variables and has limitations where the order of events is critical.

3.3.4 Observed variables and their accuracy

The Cfengine anomaly detection system is still in its early experimental stages, so only a few key variables are used to monitor the current state of a host and thus detect any anomalies in the system's behaviour. These variables are used with the two dimensional time-series and the Leap detection test to detect and classify anomalies. The current Cfengine variables can be categorized as shown in Figure 3.3

For each anomaly detected, variables consists of:

- value
- average
- standard deviation

having values that were true at the time of the anomaly's occurrance.

3.3.5 Towards variables classification

The present work will develop an ontology for current Cfengine variables as categorized in Figure 3.3. In the following section we try to describe the important aspects of each group of variables that we are interested in.

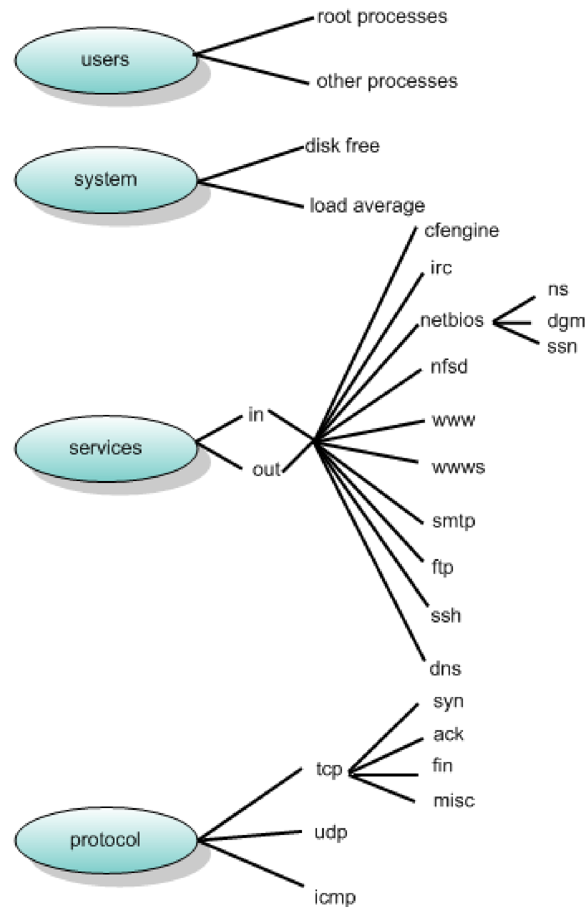

Figure 3.3: Cfengine variables

Time

For both tests, Two dimensional time-series and Leap Detection Test, the parameter "time" is important, and is considered differently. With the 2DTS test, the initial training period takes between 6 and 8 weeks. During this period, a *normal* profile of a host is developed, after which a current variable is compared to an average of equivalent earlier times. The mathematical engine in cfenvd [6,8] is used for detecting anomalies in periodic data hence it seems appropriate to use a time-period in which the data shows a type of periodicity. The periodic behavior of collected data is argued to be attributed by the environment in which a variable is observed. A single period of one week is chosen, and Cfenvd defines classes for cfagent depending on the amount of difference of the current data point from the learned average behavior for the given time slot in the period. For example, if the current data is measured at 10:00 am on Monday, it will be compared to an average of past data measured on Mondays 10:00 am time slot.

The periodic analysis of data has significance in this particular approach and sufficient work has been done to justify it[6]. We think that the time at which an event is

detected should be considered in more detail because a host's behaviour may change depending on the time of the day or week. The behaviour of hosts have been shown to follow some periodic patterns, indicating more activities for certain services during working hours than non-working hours, as well as during weekdays versus weekends as shown in Figure **??**. We see clearly

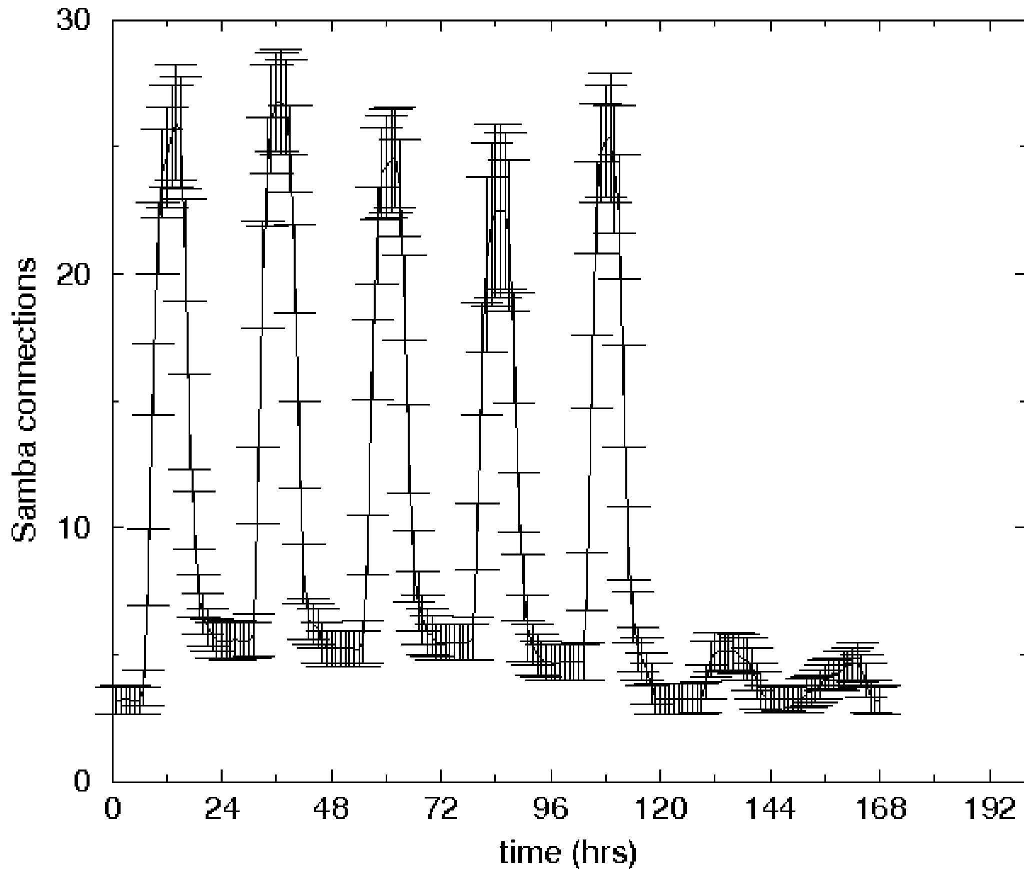

Figure 3.4: Weekly Samba file sharing service for windows and unix systems. Note how the peak for number of connections to a Samba server coincide with working hours, during weekdays.

The local time slices that has significant meaning should be considered too. For example, if a known resource intensive task (e.g backup, upgrade, etc) is scheduled for a particular timeslot, anomalies that might be detected at this particular timeslot, that we know are related to this activity should be considered as uninteresting. Questions like, when did an event occur? Was it during weekdays or weekends? Was it during working hours (when you expect more activities) or non-working hours? are relevant in filtering of events.

With LTD, the computation and detection mechanism uses only a fixed size sliding window of data and has no "memory"of past data. The issue of *time* is not as critical

in LDT in the same way as in 2DTS. For one thing, this approach does not asssume long-time periodic behaviour but only a sliding window of 25 minutes is considered in the determination of an anomaly. However, we think that the *time* parameter is similarly important for filtering interesting events.

We think that, the *time* at which a particular event occurred should be considered in the determination of the nature of anomaly or in explaining the detected anomalous behaviour. We think *time* should be incorporated in the description of the proposed Ontology to assist in the correlation of events detected within and between hosts in a distributed environment. The later case may be more applicable for distributed anomaly detection within a network, but this case will not be considered in the current work due to time constraints.

An important question here that is related to these two tests is: Can Ontology of collected events be used to identify more interesting events from the rest? How effective can Ontology be in filtering, correlating and reason on these events such that more knowledge or insights about what is really happening is obtained. These are what we plan to test in the present work. The idea is to develop an Ontology of all possible events as collected by Cfengine and tested by the two tests, and compare the effectiveness of Ontology in filtering of events in comparison to the 2DTS and LTD tests.

System

The process of detecting anomalies in system behavior requires first defining the variables to be monitored. For each variable or group of related variables, one must also define the criteria for suspicious behavior such as setting thresholds that are compared to current values obtained from statistical analysis. In the present work, this takes the form of ranges of values considered to represent abnormality, for example the measured values are classified in units of standard deviation.

Many variables are possible, but we believe that to start with, the few variables used by Cfengine anomaly detection system can be useful for detecting anomalous system behavior as shown so far. As indicated in Figure 3.3, the variables in this category that we are concerned with are: number of root processes; number of other processes; the amount of free disk and CPU load average.

As a preliminary step in the development of the said ontology, we intend to look at (but not limited to) the following aspects of *system* monitoring as related to the mentioned variables:

- typical general behavior/ characteristics of each variable
- how is the variation of each variable?
- how is the use of these variables related to others?
- which combination of the occurrence of variables might be significant

We believe that a detailed description of these aspects of system variables will help in the process of ontology development.

Services

As indicated in Figure 3.3, Cfengine considers only a number of common network services. The incoming and outgoing socket counts to and from these network services are monitored for anomalies. We feel it is important to analyze in detail the expected normal characteristics and behaviour of users and hosts (server or client) with respect to these services in order to be able to define and describe them in the ontology. For example, we need to identify interesting relations between client and server for each service, which together with hosts and users profiles can shed more light in understanding the events that are detected. Specifically we intend to look at:

- probable causes for service behavior changes
- how service variables relates to other variables like rootprocs, loadavg, etc
- typical behaviour of each service to identify deviations

Role of a Host

In our setup, a host can be workstation or a server. Our experimental network consists of two workstations and two servers. Each host in the network has specific roles, for example servers provides certain services to its clients. We think a role of a host in the network plays an important part in deciding whether an event is interesting or not. Experience shows that every host has a precise behaviour in terms of services running on it, which in general does not change significantly over time, at least in the short time-frame. This means, it is important to identify:

- which services are running in a host
- how a host's behaviour affects these services and vice versa
- factors affecting service availability in a host
- which variables shows the availability of a service?

Additionally, we need to identify *necessary* versus *supporting* variables in relation to *service* or *system* variables in a host.

To summarize, the present work is concerned mainly with ontology-based approach, to recognize non-trivial and interesting events, thus essentially filtering and reducing the amount of events that requires further analysis. We envisage to harvest the power and utility of the ontology in expressing the relationships between collected data and make use of its reasoning capability and that of inference engine to filter the interesting events from the rest.

3.4 Other Anomaly Detection Systems

IDES and NIDES were the first functional anomaly detection systems using statistical procedures to create statistical profiles that describe the normal behavior of the users

and system components. Usually the stored profiles are constantly being updated in order to reflect changes in user or system behavior. If there is a severe deviation from the normal profile, the system reports a security violation[8]. The Intrusion Detection Expert System (IDES) developed at SRI performed Intrusion Detection by creating statistical profiles for users and noting unusual departures from normal profiles [9]. IDES keeps statistics for each user according to specific Intrusion Detection measures, such as the number of files created and deleted each day. These statistics form the statistical profile of each user. The profiles are periodically updated to include the most recent changes to the user's profile. Therefore, this technique is adaptive with changing user profiles. However, it is also susceptible to a user slowly changing his or her profile to include possibly intrusive activities[10].

EMERALD (Event Monitoring Enabling Responses to Anomalous Live Disturbances) [16] environment is a distributed scalable tool suite for anomaly and misuse detection and subsequent analysis of the behavior of systems and networks. The development of EMERALD drew on experiences the authors had with IDES and NIDES.

Change and Anomaly Detection (ChAD), is a work by Stottler Henke Associates, Inc. (SHAI) for anomaly detection and fault prediction. ChAD models the normal behaviour of a system and detects sudden changes like those caused by a denial of service attack in a computer network, as well as changes occuring over time such as wear and tear of hardware components. Additionaly, ChAD system may be used to create models of a variety of "normal" operating conditions, for example peak traffic periods, such that it can segment these periods in order to detect correctly, anomalous behaviors.

Chapter 4

Ontology development

This chapter describes the methodology used to structure an ontology for host based anomaly detection. In order to give concepts and their relationships meaning, we must relate an underlying model of our anomaly detection to the ontological representation.

Ontology development is nontrivial, one reason for this difficulty is that ontologies are formal models of human domain knowledge [25]. Human knowledge is often tacit and hard to describe in formal models, and there is also no single correct mapping of knowledge into discrete structures. Although some rules of thumb[1] exist that facilitate selected ontology design tasks, for example as mentioned in [25, 26], there are hardly any comprehensive ontology development methodologies that has been agreed upon and in use today. Several suggestions has emerged from the experiences of ontology developers such as TOVE (based on experiences in the development of Toronto Virtual Enterprise ontology), Methontology [27], and Enterprise Model Approach (based on experiences in the development of the Enterprise ontology [23]). Most often, terminology used in the domain of interest is gathered and organized into a taxonomy, from which key concepts are identified and related to create an ontology. We have divided the work of ontology development in two main phases namely, Knowledge capture which is described in section 4.1, and Implementation is covered in section 4.2.

4.1 Approach to behaviour discovery in events

This phase is referred to as the knowledge capture phase whereby we look for the information embedded in events, and collect terminologies which can be used to formulate *concepts*, *properties* and *relations* between them, with the ultimate goal of combining these into an ontology for the purpose of interesting events identification. We believe that we can relate hosts behaviour if we know relationships between them, which can tell us more about the events captured in the network.

[1]There is no one correct way to model a domain, there are always viable alternatives. The best solution almost always depends on the application one has in mind and the anticipated extensions

4.1.1 Terminology identification

Since with Anomaly detection we want to identify anomalous behaviour, we think that we should explore factors affecting a host's behaviour, which can be external (e.g Users, remote network and application services) or internal (e.g processes running in a host, CPU load, role of a host etc). We think such information is crucial in modeling the "normal" state of a host, from which anomalous behaviour can be recognized. Based on the purpose and domain of the ontology, the terminology collected where from events and other related concepts that were thought to be part of anomaly detection process with Cfengine. Figure 4.1 shows the main concepts we came up with at this stage.

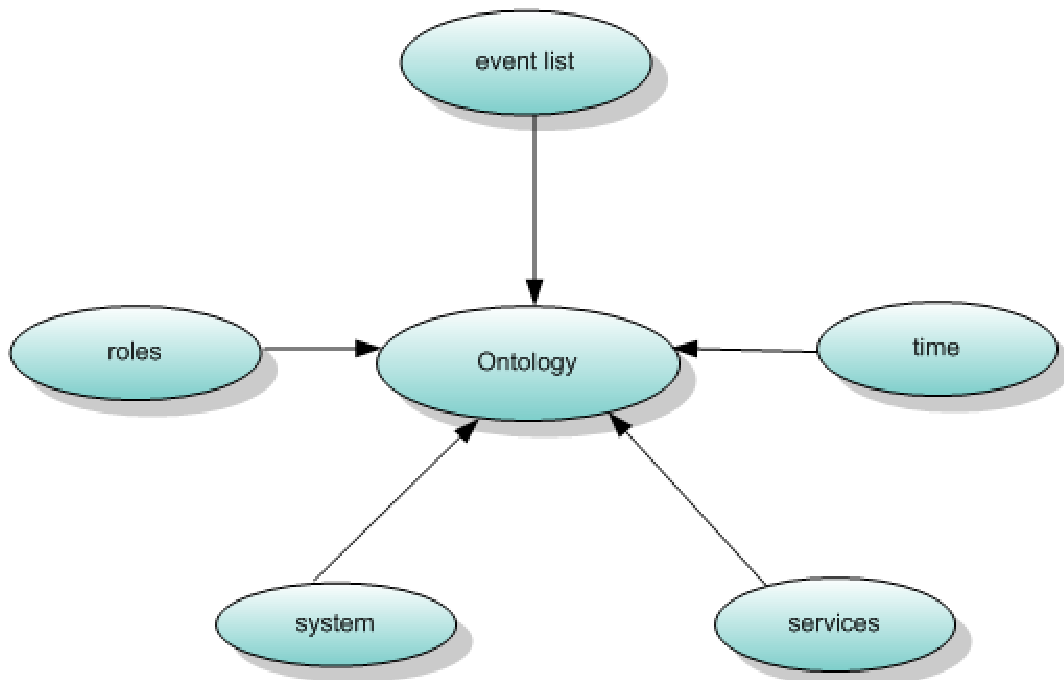

Figure 4.1: Main concepts related to events

The concepts shown by Figure 4.1 were chosen to be the main building blocks of the proposed ontology and they saved as a starting point in the thinking process of how to formulate, organize and relate concepts inherent in anomaly detection.

4.1.2 Formulating concepts relations

The process of identifying key concepts and relations between as an input to the ontology model was done iteratively, starting from a broader perspective to a more specific domain. In trying to formulate the relations between concepts, we thought the best way to represent effectively the knowledge inherent in events is to look at generic terms and concepts used in Anomaly Detection. This resulted to a number of terms

such as Observation; Measurement; History; Statistics; Uncertainty; Variable and Time as shown in Figure 4.2.

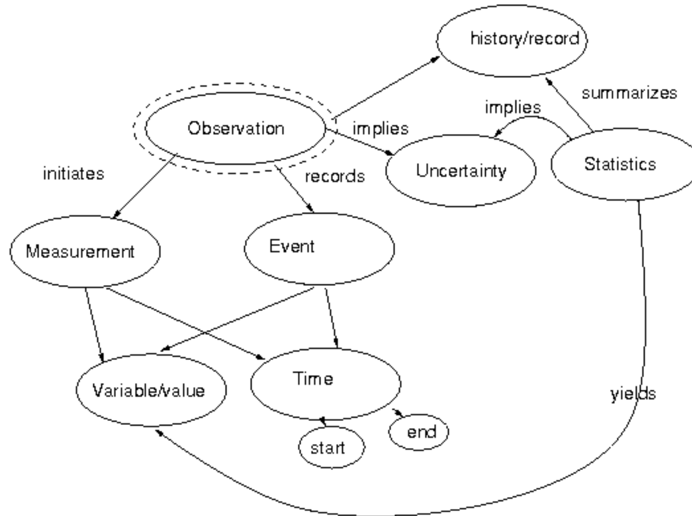

Figure 4.2: Taxonomy of observations with some example relations.

The taxonomy of observations in Figure 4.2 covers a wide scope of generic network/host monitoring, measurements and statistics. We saw the need to narrow the scope and try to break the domain of interest into smaller sub-taxonomies such as that covering variables and related concepts, shown in Figure 4.3.

Zooming into an area of interest makes it easy to formulate relations between immediate concepts, which can be expanded further iteratively until the whole domain of interest has been covered.

From the insights obtained from looking at Variables scope, the list of all possible events from Cfengine anomaly detection system was once again analyzed, and events were grouped into taxonomic hierarchy, e.g network services, processes, whether initiated by a user or another program, and if it is a user, is it a privileged user or not, as shown in Figure 4.4.

We refer to Figure 4.2, Figure 4.3 and Figure 4.4 as taxonomies rather than ontologies because they do not represent all possible relations and their constraints, about concepts, neither graphically nor logically as ontology does.

4.2 Implementation: Building the Ontology

It has been mentioned previously that, there is no 'correct' way of building ontologies. Since no comprehensive methodology for ontology development has been standardized at this point in time, this section presents heuristic methodological approaches

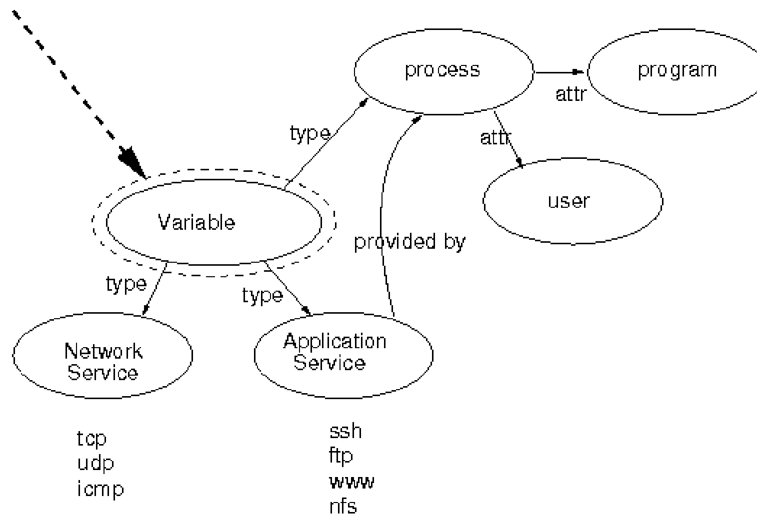

Figure 4.3: Sub-Ontology for variables

for ontology development adopted in the present work. This approach consists of four main phases, namely:

1. Defining the purpose, scope and requirements for the ontology.
2. Creating a conceptual model underlying the ontology using ontological modeling primitives (e.g, concepts, properties and relations).
3. Conveying the conceptual model in a representation formalism that allows ontology to be read by machines.
4. Evaluating the built ontology.

These steps will be used to describe the ontology development process done in this work in the subsequent sections.

4.2.1 Determining purpose, scope and requirements

The purpose for the ontology was set forth during Project design phase. As mentioned previously, the proposed ontology is designed to support reasoning for event filtering through the use of Description Logic based languages like OWL.

In terms of scope, the proposed ontology will be confined to the 'events' collected by Cfengine (cfenvd, LDT and cfagent) instead of an Anomaly Detection domain as a whole.

Requirements of an ontology are a means to achieve the purpose of the ontology. Since the conceptual model is meant to be an explanation of the domain of interest, and a tool through which the requirements of the ontology are met, it is important

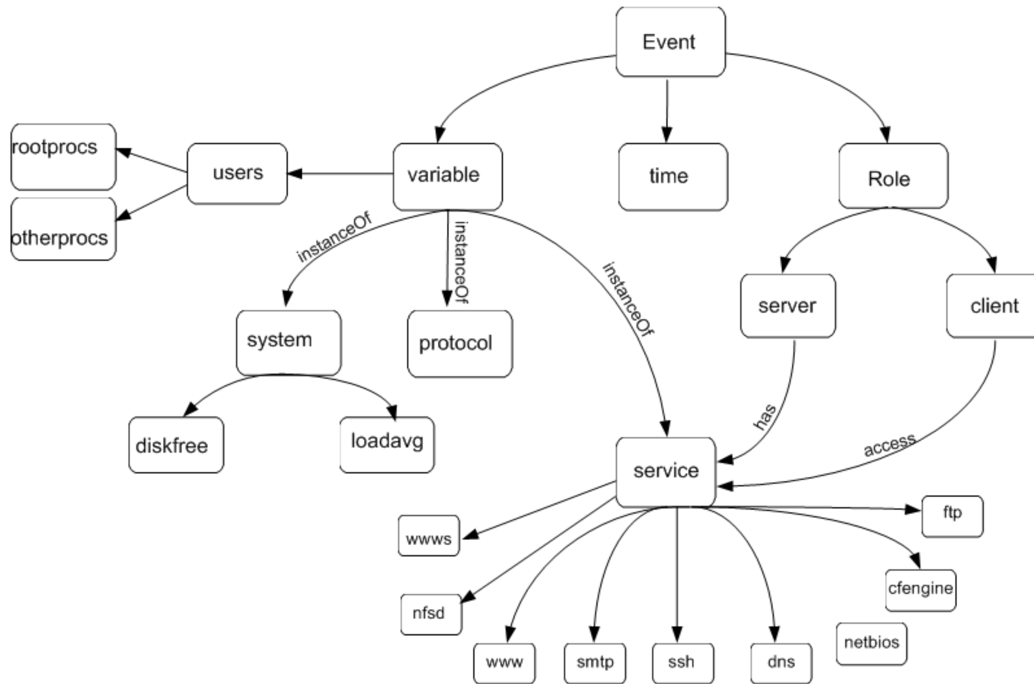

Figure 4.4: Taxonomy of events

that the terminology as well as the knowledge structures be represented correctly. We identified the key requirement of our ontology to be that, convey as much as possible the 'normal' state of the system, in the ontology by describing the events, such that they reflect the normal system behaviour, in the hope that the reasoner will be able to infer an odd (abnormal) event or unusual combination of events.

4.2.2 Conceptual model

The first step in creating an ontology conceptual model, is often to create a *taxonomy*. It is important at this point to define what we mean by a taxonomy which is used interchangeably with the term *hierarchy* in this text. Generally, taxonomy refers to the practice of hierarchically classifying things or concepts according to some taxonomic scheme (how they are related). Taxonomy is often represented in a tree-like structure (hence the common use of the term hierarchy). We have observed the tendency of some people to use the terms taxonomy and ontology interchangeably, so we feel there is a need to differentiate these terms, especially as related to this work.

Whereas a taxonomy refers to a hierarchy created by grouping entities according to their inherent data ('a kind of' relation) in the hierarchy, the ontology in addition to that, describes further these entities, stating more properties and relationships between them. Ontology specifies richer semantic relations between entities apart from is-a relation. We can conclude that a taxonomy can be part of an ontology but not vice versa. The following statements shows further relationships between these terms.

```
Vocabulary + Structure = Taxonomy
Taxonomy + Relations, Restrictions and Rules = Ontology
Ontology + Instances = Knowledge Base
```

Several iterations of manually redefining concepts and arranging them into a hierarchical taxonomy (superclass/subclass relationships), resulted to a taxonomy shown by Figure 4.5, which became the underlying model of the built ontology.

Note that we refer to Figure 4.5 as ”taxonomy”. This is because at this point, the class and instance properties and constraints are not set yet to make it an ontology.

4.2.3 Semantic relationships

In anomaly detection we are often concerned with the impact of events on the system. To characterize this, the main relationship of interest is *causality* of one event with respect to another. This is made more complicated because Cfengine classifies events levels into “buckets” of statistical deviation sizes. So, although it might be true that an increase in user logins causes an increase in the number of processes on a system in a detailed view, it is not necessarily true that a move from one measurement class of user logins to another will imply a corresponding move to the next measurement class for processes. This depends on how full each “bucket” is to begin with. This is the basic difficulty with all statistical analysis. Figure4.8 shows possible causal relationships for some of cfenvd source variables. We want to identify all possible relationships between variables making up an event, and in that way represent the knowledge we have about “normal” system state as represented by the behaviour and characteristics of such variables. We propose to adopt the open-world assumption[2] reasoning supported by Protege and OWL, to account for the limitation in representing all possible relations between events components. This helps to validate the ontology and allow for future extension. More discussion about this is given in the following section.

We would like to understand relationships between these events for two main reasons:

- If two events always occur together, then we could collapse them into a single event to avoid redundant reporting.
- If certain clusters of events have special significance then we could again collapse these into new meta-level events by aggregation and provide a more meaningful diagnosis. (This is essentially what rule-based detection systems do at a lower level).

As we have noticed from the ongoing discussion, *relationships* that are specified through the use of properties and their constraints are very important part of class definitions and description which is the way ontology represents knowledge existing in the domain of interest. In addition, the relationships need to be specified in such a way that they are understandable to both humans and machines.

[2]Open-world assumption means that, just because something is not stated to be true we can not assume it to be false.

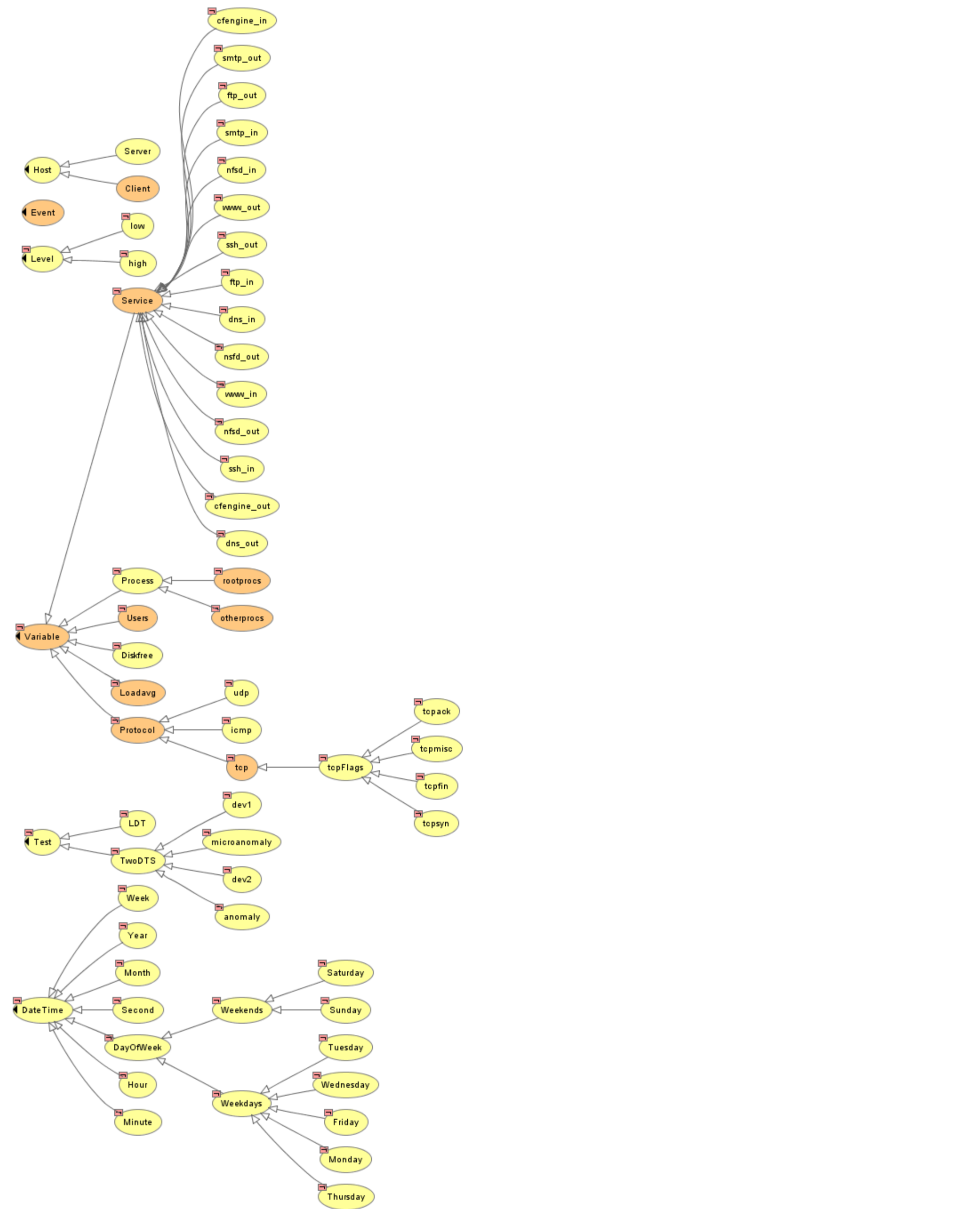

Figure 4.5: Taxonomy of events

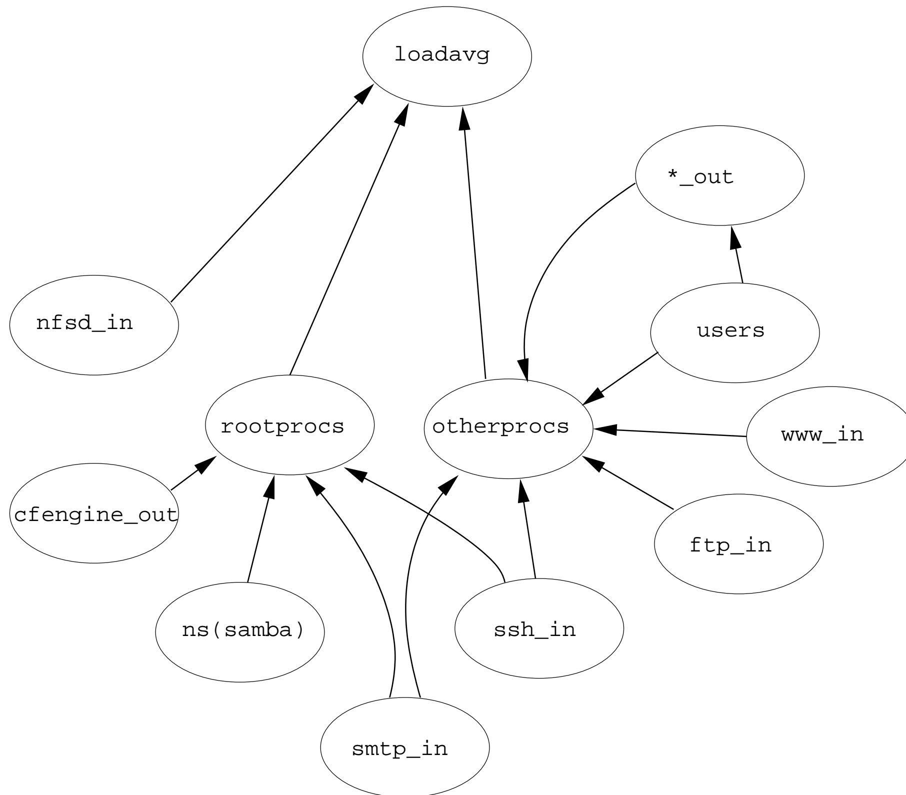

Figure 4.6: Cfenvd base class source variables with possible causal relationships. These are not necessarily reflected faithfully in the approximation buckets, nor are they necessarily always true.

In Protégé -OWL, this is done using constructs shown in Figure 4.6. However, we think semantically richer relationships can be described using additional constructors such as those shown in Figure 4.1.

4.2.4 Relationships Identification

The understanding of the design and behaviour of the Unix system are important for the semantic modeling of our ontology. We need to understand the behaviours associated with the variables that are measured. In Cfengine measurements by cfenvd lead to periodically evaluated events within the context of cfagent.

Today it is common to speak of network services and application services, where network services are the low levels of the OSI model, e.g. TCP/IP (what Unix refers to as "protocols"), and application level services are higher levels such as www, FTP, etc (what Unix refers to as "services" or reserved ports). From this point on, we will adopt the Unix naming scheme of network services and application services as *protocols* and *services* respectively, when we refer to event components in the ontology.

A service in Unix is something that is listed in the /etc/services file. It consists essentially of a portnumber and transport layer protocol type (listed in /etc/protocols).

Forward	Inverse
Is equivalent to ($\equiv$)	Is not equivalent to ($\not\equiv$)
Is approximately equal to ($\simeq$)	Is not approx. equal to ($\not\simeq$)
Is a superset of $\supset$	Is a subset of $\subset$
Implies ($\Rightarrow$)	Is implied by ($\Leftarrow$)
Leads to ($\rightarrow$)	Follows from $\leftarrow$

Table 4.1: Additional constructors and their inverses that can be used to specify relations. Note that these are not part of Protégé built-in constructors.

Services are associated with an addressable device at some point, e.g. a network interface with IP address, but this is abstracted away.

Services are normally provided by server-processes, often called "daemons". A few services (e.g. some implementations of NFS) are written as kernel modules but these are invisible to Cfengine so we shall ignore them. All measurable activity on a Unix system must be associated with a process, and every process has a *user* as an attribute (i.e. processes "belong to" users). We can forget about kernel activity for the most part and think of every measurable event as being associated with a process in the process table.

We distinguish between two types of application-service: those that imply a user login to an account on the machine (e.g. ssh, ftp), and those which are anonymous services (e.g. www, ftp).

A user login spawns a shell process from the init process. This registers as a new shell for the user, thus a login implies an increase in processes and possibly an increase in the number of users (a single user can be logged in many times, but the measurement only counts this once).

Most services are serviced by "daemons" that run under a certain user's credentials. This includes user-logins by terminal which are handled by shells spawned by the init process. Most services are simply run either as the root user or under a special username for the service, e.g. the web service runs as the "wwwrun" user on most GNU/Linux systems today. The ftp service is unusual as it allows both named and anonymous (guest) users. Anonymous users are assigned a special local username also however, which is usually called "nobody" and falls into the category of "otherprocs". There is no clear relationship between the number of logins, users and the number of connections however, since a single user can log in multiple times.

Event class	Relationship	Event Class
ssh connection	implies	login
ftp connection	might imply	login
imap connection	implies	login
login	implies	process

Of all the services that Cfengine monitors by default, only smtp has a relay function, i.e. this is the only service in which there could be a connection between incoming and outgoing connections.

Let us define the symbols:

Meaning	Symbol
Always causes	$\Rightarrow$
Usually causes (likely)	$\xrightarrow{\surd}$
Might cause (unlikely)	$\xrightarrow{?}$
Rarely causes (possible)	$\xrightarrow{??}$
Never causes	$\not\rightarrow$

Variable	Forward	Variable	C
$*procs$ Increase	$\xrightarrow{\surd}$	$loadavg$ Increase	0
$\left.\begin{matrix} ssh_in \\ web_in \\ ftp_in \\ smtp_in \end{matrix}\right\}$ Increase	$\xrightarrow{\surd}$	$processes$ Increase	1
$rootprocs$ Increase	$\xrightarrow{??}$	$otherprocs$ Increase	2
$users$ Increase	$\Rightarrow$	$*procs$ Increase	3
$users$ Increase	$\xrightarrow{?}$	$*_out$ Increase	4
$users$ Decrease	$\xrightarrow{?}$	$*procs$ Decrease	5
$users$ Decrease	$\xrightarrow{?}$	$*_out$ Decrease	6
$\left.\begin{matrix} ssh_in \\ web_in \\ ftp_in \\ smtp_in \end{matrix}\right\}$ Increase	$\Rightarrow$	tcp_syn_in Increase	7
$\left.\begin{matrix} ssh_in \\ web_in \\ ftp_in \\ smtp_in \end{matrix}\right\}$ Decrease	$\xrightarrow{\surd}$	tcp_fin Increase	8
$(smtp_in \wedge smtp_out)$	$\xrightarrow{?}$	Mail relaying	9
$(smtp_in \wedge smtp_out)$ Increase	$\xrightarrow{\surd}$	$*proc$ Increase	10
$(smtp_in \wedge smtp_out)$ Increase	$\xrightarrow{??}$	$*proc$ Spam attack	10
Zero $*_in$	$\xrightarrow{\surd}$	Service * not functional	11
$users = system_lower_limit$	$\Rightarrow$	no users logged on	12

Table 4.2: Example relationships that describe the underlying system and provide semantics for the ontology.

It is worthwhile to note that, it is not straightforward to translate the changes

shown above into Cfengine statistical classifications, since an increase from one broad class to another implies much greater uncertainty.

LDT Classifier	Forward	LDT Classifier	C
$*procs$ Change	$\xrightarrow{\surd}$	$loadavg$ Change	0
$\left.\begin{array}{l} ssh_in \\ web_in \\ ftp_in \\ smtp_in \end{array}\right\}$ Change	$\xrightarrow{\surd}$	$processes$ Change	1
$rootprocs$ Change	$\xrightarrow{??}$	$otherprocs$ Change	2
$users$ Change	$\Rightarrow$	$*procs$ Change	3
$users$ Change	$\xrightarrow{?}$	$*_out$ Change	4
$users$ Change	$\xrightarrow{?}$	$*procs$ Change	5
$users$ Change	$\xrightarrow{?}$	$*_out$ Change	6
$\left.\begin{array}{l} ssh_in \\ web_in \\ ftp_in \\ smtp_in \end{array}\right\}$ Change	$\Rightarrow$	tcp_syn_in Change	7
$\left.\begin{array}{l} ssh_in \\ web_in \\ ftp_in \\ smtp_in \end{array}\right\}$ Change	$\xrightarrow{\surd}$	tcp_fin Change	8
$(smtp_in \wedge smtp_out)$ Change	$\xrightarrow{\surd}$	$*proc$ Change	10

Table 4.3: LDT relationships more closely represent the underlying variables in Cfengine, but they represent sudden changes of gradient, not absolute values.

For Cfengine classes, an increase means a move from one class to another, e.g. from *dev*1 to *dev*2.

Working together, the LDT and 2DTS Cfengine classes cover the essential features of the underlying behavioural variable. However, by absorbing the statistical fluctuations in values so as to avoid false-positives, the causality of the relationships is obscured. This means that it is hard to use a logical approach to classify them further. The least we can do is to represent the possible behaviours we can specify. The process of identifying class properties and relations between them was similarly iterative. We started by generally thinking and elaborating the properties, normal behaviour and how variables are related. We then tried to express the relations we came up with progressively, as shown by Table 4.2, Table 4.3, and Table 4.4. However we found that it was not possible to convey and relate the changes shown by events correctly and completely using Protégé and the OWL Plugin the ontology editing tool that was used in this work. Instead we formulated relations that only describes the events without

incorporating the changes. Table 4.5 shows a list of some Object properties used to describe concept while developing the ontology.

4.2.5 Implementing the ontology on Protégé

The reader is referred to relevant sections in chapter 2 for a detailed description of ontology components.

In developing the ontology in the present work, we incorporated the relationships from the previous discussion as well as descriptions about variables and concepts in the event domain. Figure 4.5 shows the asserted ontological hierarchy resulting from ontology modeling process described in section 4.2.

4.2.6 Some design decisions made

As mentioned previously, there is no one way of developing an ontology. The ontology development process presented different options for concepts classification, with no obvious criteria for selection of one option over another. The following are an example of some design decisions we had to make.

- Host to be in the same level as Event in the class hierarchy rather than as a subclass of Event. This is because all concepts that are subclasses of Event are "kind of" (Is-A /superclass-subclass relation) events, which means they have individuals belonging to both classes. From our list of events, Host and its subclasses (Client and Server) are do not form part of any event but rather events are collected from Hosts.

- Out ontology consists of 3 main superclasses, namely Host, Event and DateTime. We have decided to make Host and Event classes not disjoint, by explicitly making Event disjoint with Datetime and DateTime disjoint with both Event and Host. This is to allow relationships that may exists between Users which is a subclass of Event and subclasses of Host, since disjoint properties are inherited down the subsumption hierarchy.

- Due to the nature of anomalies, (unexpected, abnormal behaviour), it is not possible to represent uncertainties associated with them using current ontology development tools, such as Protégé 2000. We have decided to represent the known knowledge about events that we have, and experiment to see how ontology can support the Reasoner to identify the interesting events from the rest.

The ontology developed consists of 6 superclasses namely, DateTime, Event, Host, Level, Test and Variable. Each of these superclasses with the exception of Event class had one or more subclasses. Since we wanted to represent how variables and entities are structured through their relationships between them, we positioned the classes in the hierarchy to reflect these relationships. We experimented with different arrangements of classes in the taxonomy while examining for any conflicts between the relations thus specified and the actual reality. At the same time we had to compare with

the inferred hierarchy produced by the reasoner to ensure there are no inconsistency. The final taxonomy is shown in Figure 4.5.

The next step in implementation was to add the identified properties and relations between classes and individuals. This is done through the use of the OWL Plugin built-in constructors, whereby named classes are *defined* and *described* using *necessary and sufficient* conditions and *necessary* conditions respectively. Table 4.6 shows OWL built-in constructors in Protégé 2000.

As mentioned previously, an OWL ontology consists of Individuals, Properties, Classes, and restrictions on properties. These basic components were used to create our ontology as shown by Figure 4.8.

Individual

Individuals, represent objects in the domain of interest. Individuals are also known as instances. In our ontology we had four Individuals namely, Cube, Nexus, Satyagraha and Rex. It is worthwhile to note that, unlike Protégé OWL does not use Unique Name Assumption, which means, two different names could actually refer to the same individual.

Properties

Properties are relations between two individuals. For example, the property hasInfluence links an individual from class *cfengine_out* (any instance of the number of this connection) to an Individual in class roortprocs. In addition, a Property may have an inverse which is another property, e.g *isInfluencedBy* is an inverse property of *hasInfluence*. Other characteristicsof properties are as explained in 2.3.6.3

Classes

An OWL class is a set that contain Individuals. Class expressions are used to state the requirements for membership of the class. The word *concept* is often used interchangeably with *class*. We say that Classes are a concrete representation of concepts. Classes may be organized into a superclass-subclass hierarchy (subsumption relationships), which is also known as a taxonomy. For example, *cfengine_out* is a subclass of *Service*

Figure 4.7 shows a representation of classes with some properties linking some individuals together.

Note the Class expressions made up for class *cfengine_out* using properties (isOwnedBy, hasInfluence), which in addition to restricting named classes (Server, Client, rootprocs), they specify un-named classes(logical). Note also the inherited unnamed sublasses of cfengine_out through inheritance shown on the lower part of the Assertion Conditions window.

Similar process was done for other classes, filling up properties and relations between classes accordingly. The disjoint for siblings[3] was set for all superclasses which

[3]classes in the same level

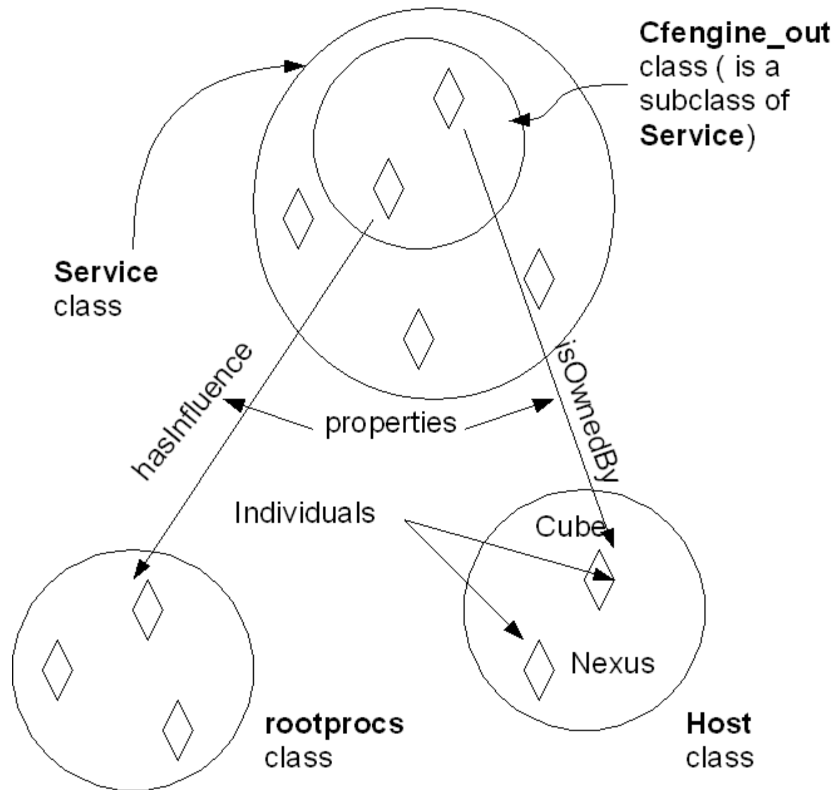

Figure 4.7: Individual, Class and Property representation

is also inherited down the hierarchy. Note that it is important not to set disjoint between classes if it is known or anticipated that some individuals might belong to both classes to avoid inconsistency.

Protégé includes the facility of converting the ontology project(.pprj) to RDF/XML as shown in Figure 4.9.

It is worth mentioning that, Figure 4.5 was created by OWLViz, which is designed to be used with the Protege OWL plugin. OWLViz enables viewing of the class hierarchies in an OWL Ontology, allowing comparison of the asserted class hierarchy and the inferred class hierarchy. However, as seen in Figure 4.5, no visual representation of relations between classes is given. More detailed view of relations is given by the Jambalaya, a plug-in created for Protégé which allows one to visualize regular Protégé and OWL knowledge bases. An example of visualization with Jambalaya is shown by Figure 5.8.

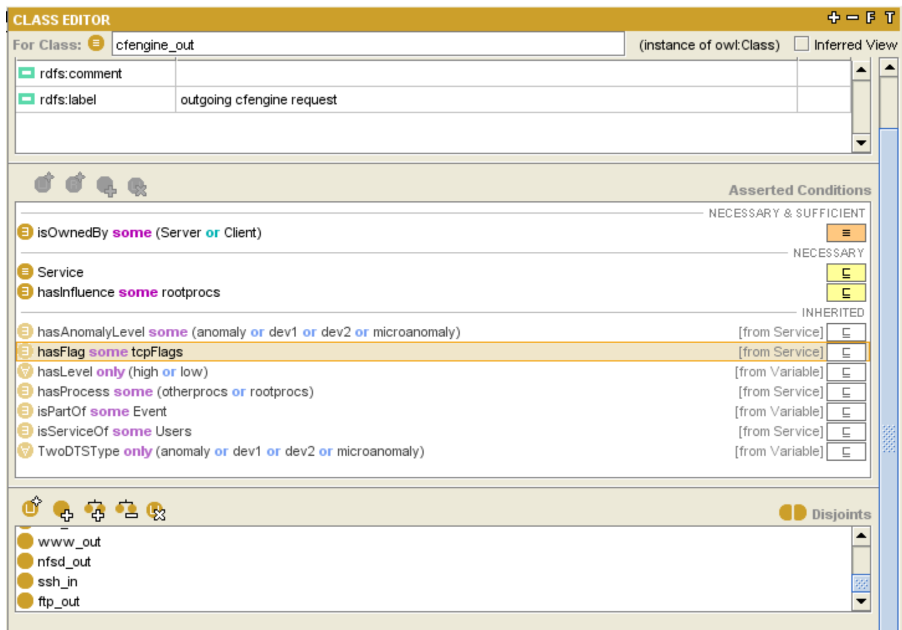

Figure 4.8: Class definitions and descriptions.

logical(machine readable content)

name of class

```
</owl:Class>
  <owl:Class rdf:ID="cfengine_out">
    <rdfs:subClassOf rdf:resource="#Service"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasInfluence"/>
        <owl:someValuesFrom rdf:resource="#rootprocs"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <owl:disjointWith rdf:resource="#cfengine_in"/>
    <owl:disjointWith rdf:resource="#dns\_in"/>
```

type

restriction on property, ***hasInfluence***

Figure 4.9: Excerpt showing class description in RDF/XML.

2DT Class	Forward	2DT Class	C
$*procs$ Increase	$\xrightarrow{\surd}$	$loadavg$ Increase	0
$\left.\begin{matrix} ssh_in \\ web_in \\ ftp_in \\ smtp_in \end{matrix}\right\}$ Increase	$\xrightarrow{?}$	$processes$ Increase	1
$rootprocs$ Increase	$\xrightarrow{??}$	$otherprocs$ Increase	2
$users$ Increase	$\xrightarrow{?}$	$*procs$ Increase	3
$users$ Increase	$\xrightarrow{?}$	$*_out$ Increase	4
$users$ Decrease	$\xrightarrow{?}$	$*procs$ Decrease	5
$users$ Decrease	$\xrightarrow{?}$	$*_out$ Decrease	6
$\left.\begin{matrix} ssh_in \\ web_in \\ ftp_in \\ smtp_in \end{matrix}\right\}$ Increase	$\xrightarrow{\surd}$	tcp_syn_in Increase	7
$\left.\begin{matrix} ssh_in \\ web_in \\ ftp_in \\ smtp_in \end{matrix}\right\}$ Decrease	$\xrightarrow{?}$	tcp_fin Increase	8
$(smtp_in \wedge smtp_out)$	$\xrightarrow{?}$	Mail relaying	9
$(smtp_in \wedge smtp_out)$ Increase	$\xrightarrow{?}$	$*proc$ Increase	10
$(smtp_in \wedge smtp_out)$ Increase	$\xrightarrow{??}$	$*proc$ Spam attack	10
Zero $*_in$	$\xrightarrow{?}$	Service * not functional	11
$users = system_lower_limit$	$\xrightarrow{?}$	no users logged on	12

Table 4.4: Cfengine statistical classes provide a coarse grained view of the absolute values of observed variables.

Object Properties
useProtocol
hasProcess
hasVariable
hasDirection
hasFlag
hasLevel
hasOwner $\longleftrightarrow$ isOwnedBy
hasService $\longleftrightarrow$ isServiceOf
hasPart $\longleftrightarrow$ isPartOf
hasInfluence $\longleftrightarrow$ isInfluencedBy

Table 4.5: A sample of properties used to describe and define events in the ontology.

OWL constructor	Symbol	Example expression in Protégé
allValuesFrom, (only)	$\forall$	$\forall$ hasLevel only (high or low)
someValuesFrom, (some)	$\exists$	$\exists$ hasPart some Loadavg
hasValue, (has)	$\ni$	$\ni$ hasPart has Users
cardinality, (exactly **n**)	$=$	$=$ hasPort has 25
minCardinality, (minimum **n**)	$\geq$	$\geq$ hasClients min 1
maxCardinality, (maximum **n**)	$\leq$	$\leq$ hasServers max 5
intersectionOf, (AND)	$\sqcap$	$\sqcap$ MozzarellaTopping and TomatoTopping
unionOf, (OR)	$\sqcup$	$\sqcup$ hasDirection only (in or out)
complimentOf, (NOT)	$\neg$	$\neg$high
oneOf, (Enumeration)	$\{\}$	$\{RexSatyagraha\}$

Table 4.6: Some synonym operators and their meaning used for constructing OWL relationships. Some class descriptions are made from the combination of these. **'n'** is the number of elements in a relation being described.

Chapter 5

Results and discussion

5.1 Evaluation process

In normal cases where rather large ontologies are developed, the work is a joint effort of ontology designer and the domain expert. The later specifies the requirements of the ontology to be developed while the former implements the requirements. The built ontology is then tested by both parties, in order to ensure the satisfaction of the user's needs. However, to the authors knowledge, there are no existing standard methodologies for Ontology Evaluation. In [25], Noy et al suggests the use of competency questions[1], which are basically for testing the knowldge base[2].This work however employed heuristic evaluation using the reasoner, RacerPro. Theoretically, we can formulate ontology testing to include the following steps:

- Consistency checking:
  This is done to ensure that none of the definition of an ontology element contradicts one another. The ontology designer must verify the consistency within ontology, to avoid the risk of logical problems when the ontology is used in a knowledge base to infer new facts.
- Ontology Validation:
  During this step the domain expert tests the ontology and estimates whether it satisfies the user's needs, in which case the ontology expert can improve the conceptualization of ontology, according to such estimations.

In addition, Protégé OWL Plugin provides a mechanism to execute a configurable list of tests, on the OWL tab of the editor. These tests are small Java programs that basically verify arbitrary conditions specified in ontology, and in case of failure, return an error message. For example, one of the predefined tests checks if a disjoint property is set between sibling classes. If any property in the ontology violates any of the predefined conditions, the system displays a warning. The editor also provides a "repair" button, which removes the source of the violation automatically. Figure

[1]Competency questions are the questions that the ontology should be able to answer
[2]an ontology populated with instances for a particular application

5.1 shows the warning from the OWL Plugin (marked 'Test Results') about missing disjoints between OWLThing(Universal Superclass) and Variable class with their respective siblings. This is not a critical error and the test can add the missing disjoints through the use of 'repair' button.

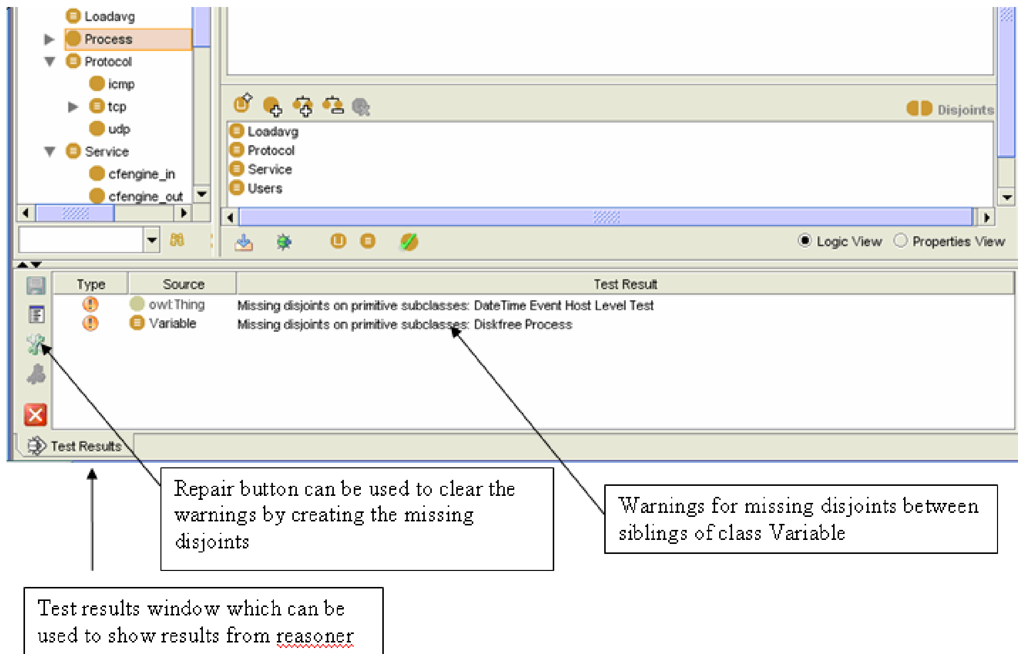

Figure 5.1: Test Results from OWL Plugin.

Use of queries is another method that can be used to evaluate ontologies. Some reasoners like RacerPro has an integrated query engine. As mentioned previously, RacerPro uses new Racer Query Language (nRQL), to submit queries about Individuals in the ontology. Protégé 2000 incorporates a SPARQL query engine which can be used to query the ontology. However meaningful querying requires a populated ontology since the queries return results about Individuals. Extensive querying using nRQL and SPARQL waere not done during the course of this work, due to time limitations. However, the use of SPARQL querying facility with OWL Plugin is shown in Figure5.2

For clarity, the query is written as:

```
SELECT ?subject ?object
WHERE {?subject rdfs:subClassOf ?object
```

The results is the list of all superclass/subclass relations in the ontology(even those which are inconsistent as shown by the red colour

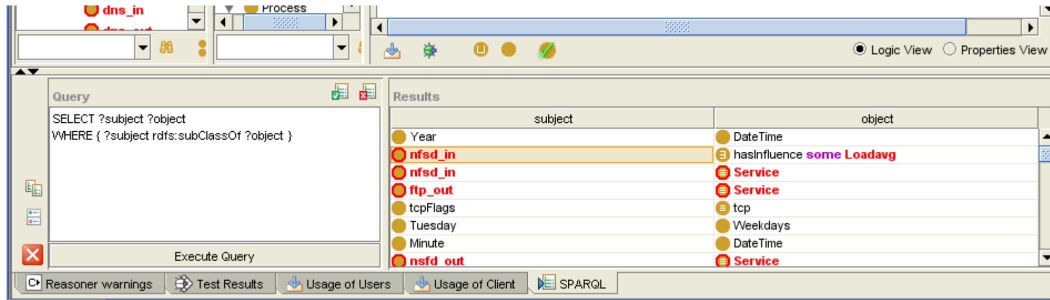

Figure 5.2: An example of a simple SPARQL query to list subject/object subsumption relation

5.2 Results

Ontology development has a continuous iterative life cycle, allowing the use of reasoners among other tools, during and after development stage. We mainly used RacerPro reasoner to check the consistency of classes within our ontology (i.e., determining whether a class can have any instances) and to compute the subsumption relations (a hierarchy which arranges the classes according to the superclass/subclass relationship).

The results of such tests are shown in the following snapshots.

Ontology classification action makes the reasoner perform consistency checks and the results can be seen from both RacerPorter and on the OWL Plugin on ProtégéḞrom Figure 5.3, the text on lines 13 and 14 (OKAY) shows that all classes are consistent that is all classes can have instances. In addition we can see a portion of the ontology taxonomy.

From the OWL tab on Protege editor, one can invoke the reasoner to perform consistency check, taxonomy classification, computation of Inferred types (for Individuals) and run ontology tests. This is shown in Figure5.4. Note the results obtained from "Run ontology tests" displayed in the Test Results window, which shows missing disjoint for the Service subclasses. There is an option for correcting this type of error using a button 'repair' on the left side of the results window.

The ontology developed with this work this far is relatively small, so no suggestions about class hierarchy was given by the reasoner as indicated in Figure 5.9. We can see that, the asserted (manually created hierarchy) and the inferred hierarchy are identical.

To further test the ontology, we created an inconsistent class "Users" by giving it two parents Client and Server which are disjoint classes as shown by Figure 5.6. See the description in the Necessary condition widget under Asserted conditions. Note also the error description in the Test Results window.

Even though it is said that necessary conditions are not used by the reasoner to infer subsumption relation between classes, we observed that describing two disjoint classes using "necessary conditions" results to inconsistencies as shown in Figure 5.7. Notice the disjoint classes Client and Server in the condition widget.

Jambalaya plug-in can be used to have a visual representation of relationships,

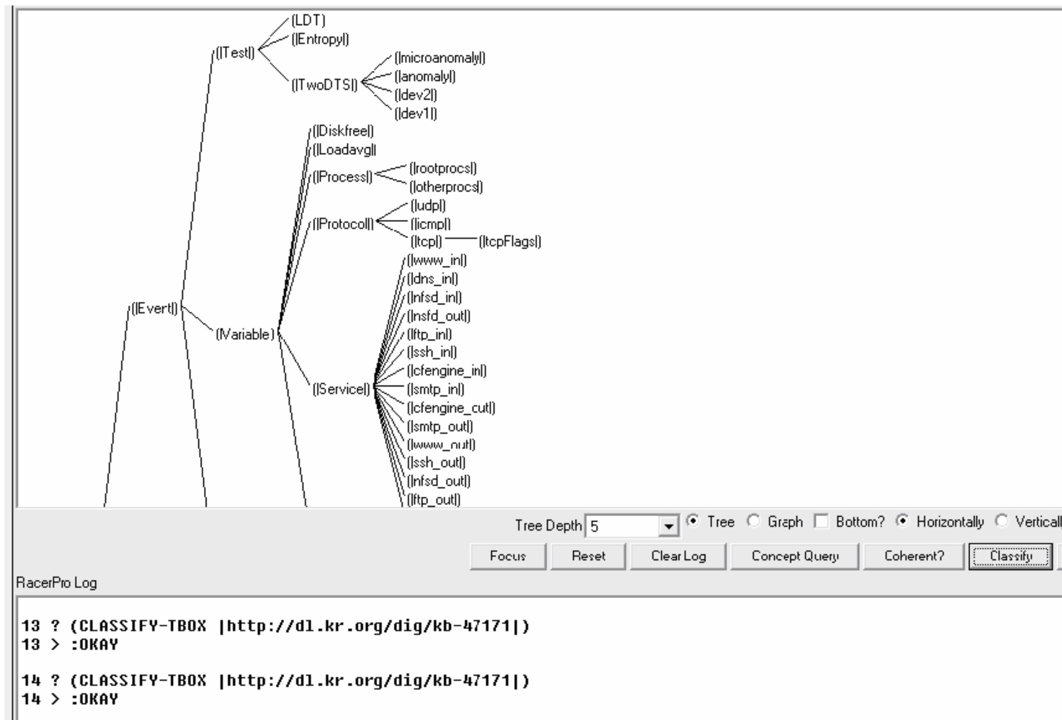

Figure 5.3: .
Results of Classifying the ontology. Note the 'classify' button functionality and the results of such action in the RacerPorter Log window indicating how these two collaborate.

restrictions and general neighbourhood of a class. An example of a view from the Jambalaya plugin is shown in Figure 5.8

The use of SPARQL querying facility is shown in Figure5.9

5.3 Alternative ontologies

We find it necessary to stress that there is no one correct way of developing ontologies, even for the same domain. This fact has been experienced during the course of this work. The process of representing the concepts comprising our domain of interest was an iterative process which resulted to several alternative representations, one of which is shown in Figure 5.13.

In Figure 5.13, we tried to represent a subset of possible types of events collected by Cfengine, and tried to relate them graphically in a is-a relation. The intention for this was to be able to identify concepts (classes) and formulate formal relations between them as a first step to represent them semantically with ontology. However, the analysis of relationships between various concepts in our domain of interest was a non-trivial task. We observed that, we needed to revisit the purpose of our ontology to determine which kind of knowledge do we need to represent with ontology, and its

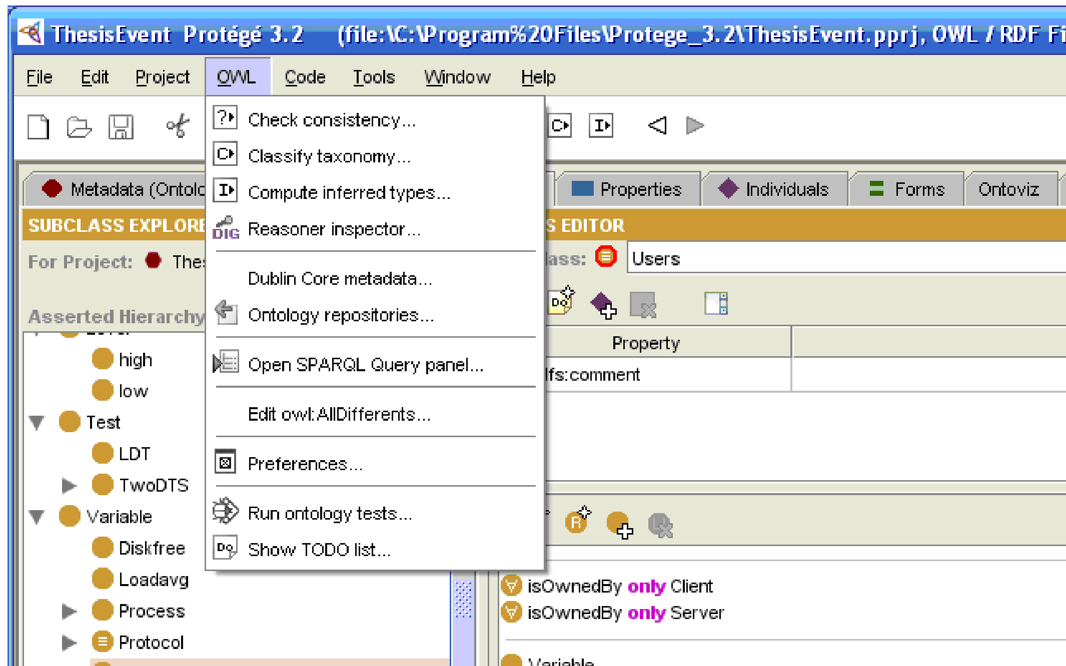

Figure 5.4: Results of Run ontology tests

scope in order to decide how detailed our relations need to be. For example in trying to relate *rootprocs* and *otherprocs* we thought of grouping these as subclasses of class *process*. We then were faced with questions such as how can we describe processes? Is it in terms of how they are started? or who owns them (system or user)? How can we relate the process thus described with the number of processes (rootprocs or otherprocs) which is basically what is collected by our anomaly detection? This line of thinking necessitated several changes to our proposed class hierarchies.

5.3.1 A promise theory representation

For a completely different approach to modeling the system, based on functional objects rather than abstract concepts or types, one can look at a promise theoretic approach. This is the description used by Cfengine itself, and it is close in idea to the Service Oriented Architecture.

Promise theory is a model of advertised behaviour, created by Burgess[6]. It deals explicitly with the advertisement of decisions that have been made. The following literature is based on a promise theory description of the Cfengine and discussions in our research group at Oslo University College.

We consider a number of agents, each with private knowledge. The agents' knowledge is "flat", it does not necessarily have a classification according to any particular model, but we assume 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 is promised.

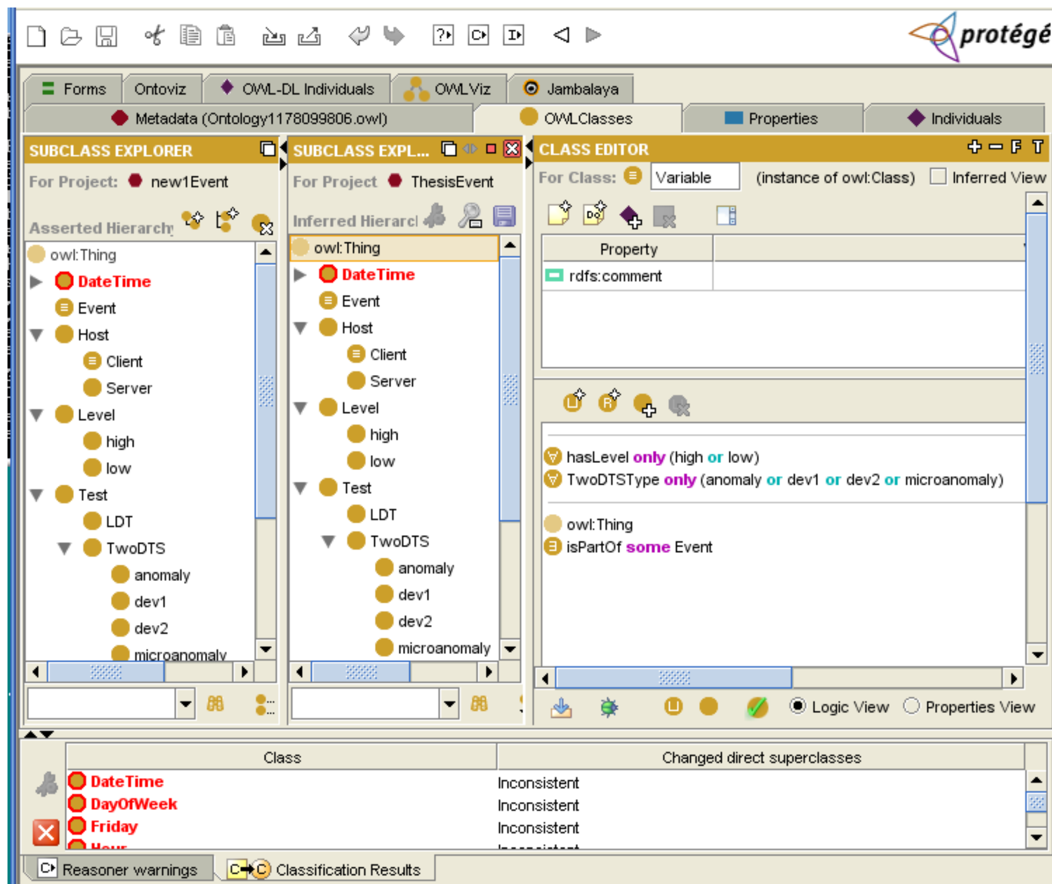

Figure 5.5: Results of taxonomy classification which produces an inferred taxonomy, as well as suggested changes to class hierarchy

A promise model is a set of promises that will lead to interactions between the agents. The behaviour 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.

While not all facts(knowledge) can be explicitly represented in an ontology (due to the nature of human knowledge), propositional facts can be promised since they are simply a kind of knowledge or agreement. 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,

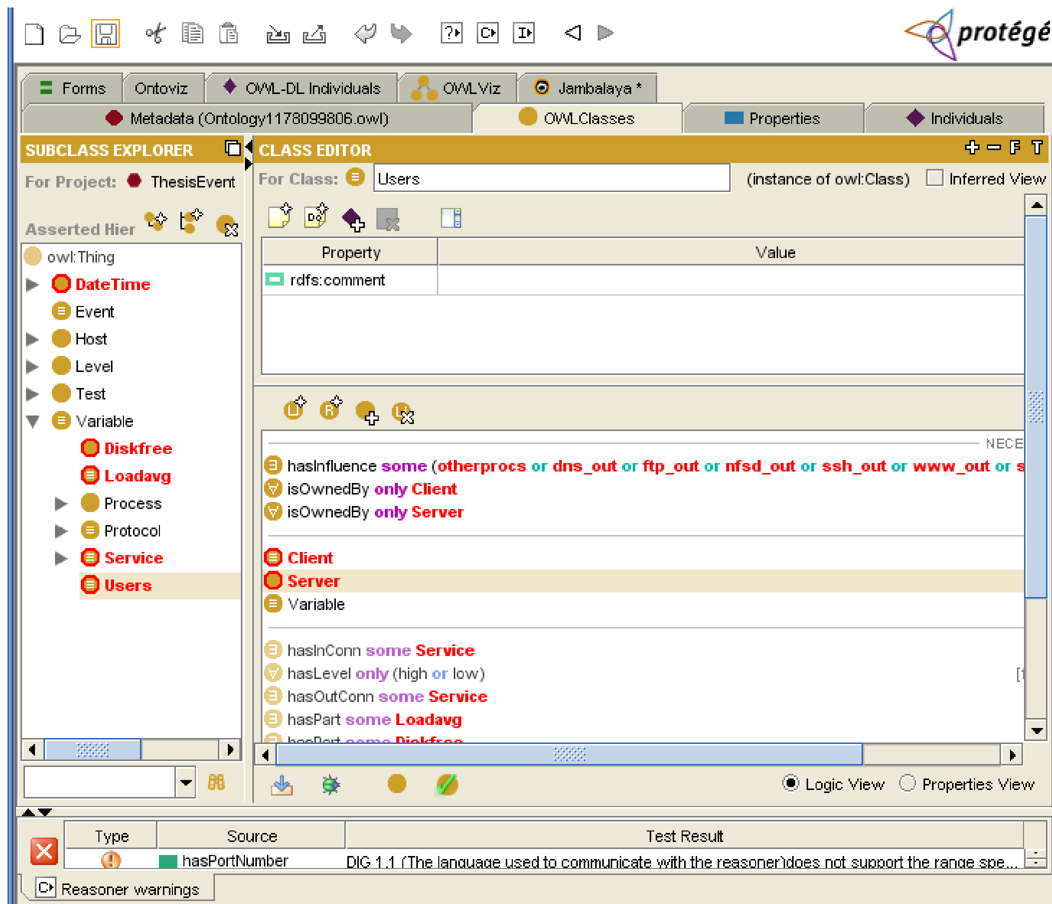

Figure 5.6: Results of consistency check

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 (see ref. [7] as an example for Object Oriented programming).

5.3.2 Promise Theory approach to modeling Anomalies

The main problem in classifying statistical anomalies using a framework of logical reasoning is the high level of uncertainty. Logic works best when uncertainty is negligible.

The lack of clear relationships between statistically aggregated data classes means that anomalies live up to their name as unclassifiable. So what is the point of looking

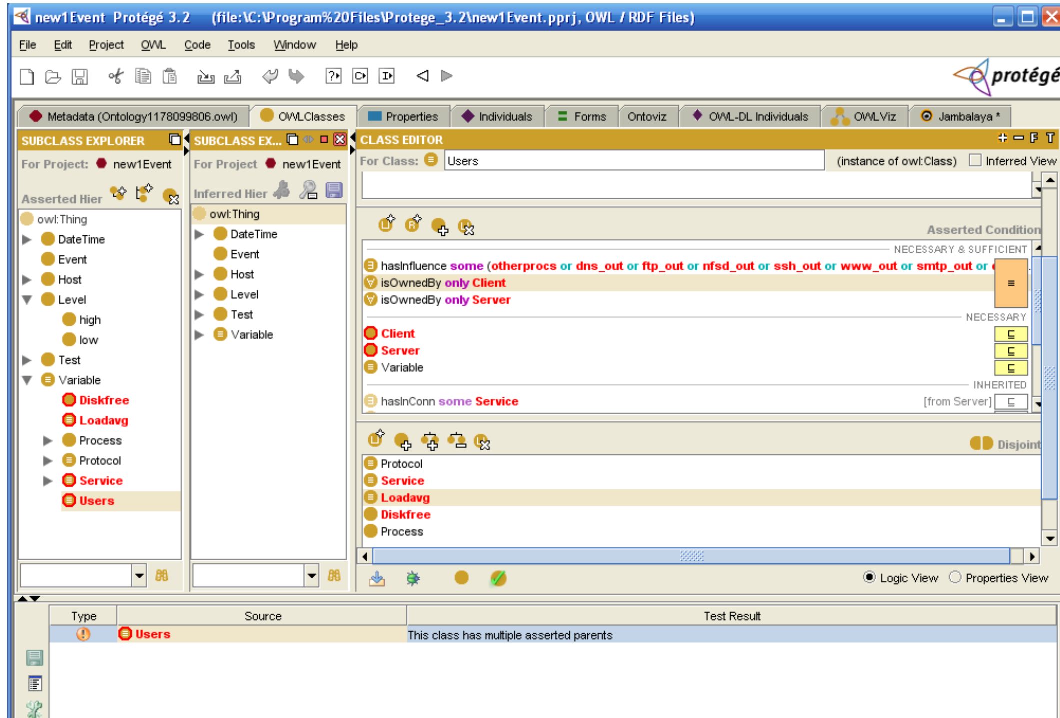

Figure 5.7: Inconsistencies caused by class description using *necessary condition*

for anomalies? A reliable signal of a significant change in a system can work as a first alarm to allow a human investigator to look more closely.

The key relationships that we can infer from the system behaviour are all things that can be written as unreliable promises, e.g.

- Promise that an increase/decrease in X leads to an increase/decrease in Y.
- Promise that X is dangerous/undesirable with probability P.

The main conclusion we can make is this in Table Figure5.1 The question is how

Variable	Forward	Variable
Large number of $*$ anomalies	$\Rightarrow$	Change in system behaviour

Table 5.1: Some general conclusions about the events.

can we best represent the knowledge obtained from the promise theory model of anomalies? Can we use ontology? We think this warrants more research and should be part of future work.

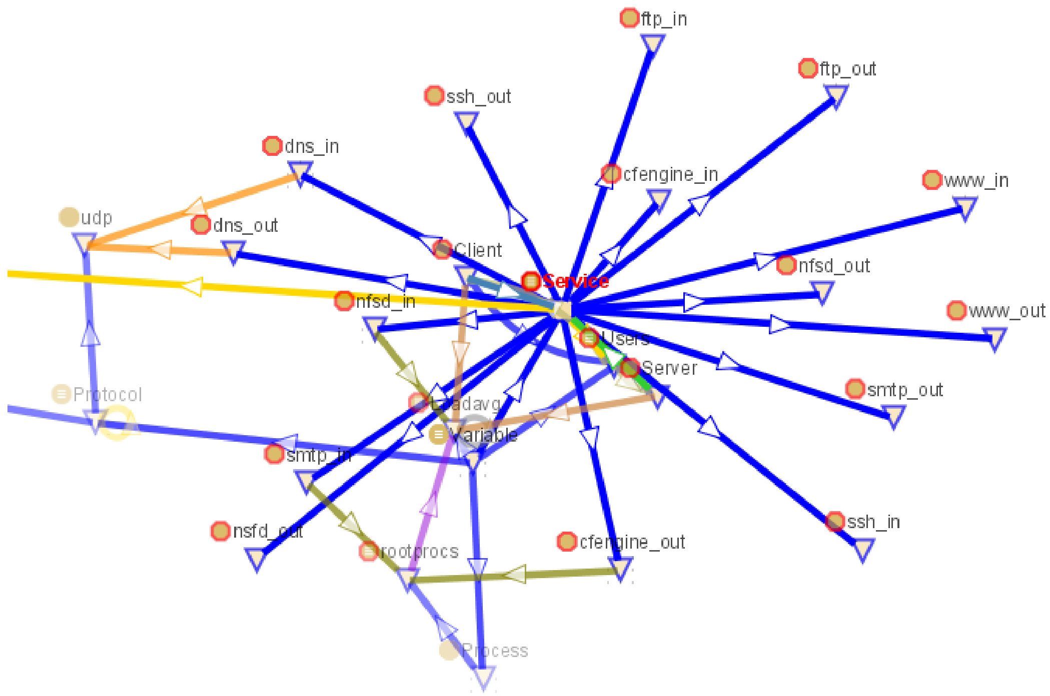

Figure 5.8: Jambalaya view of a class **Service** neighbourhood. Navigating by mouse on colour-coded links shows the type of relations between classes including inferred relations.

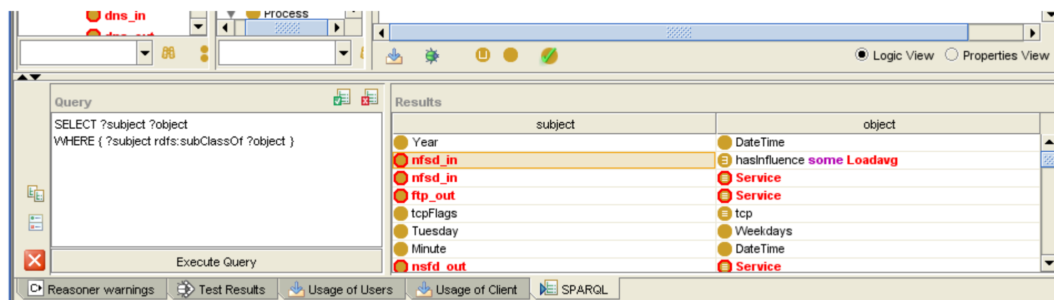

Figure 5.9: An example of a simple SPARQL query to list subject/object subsumption relation

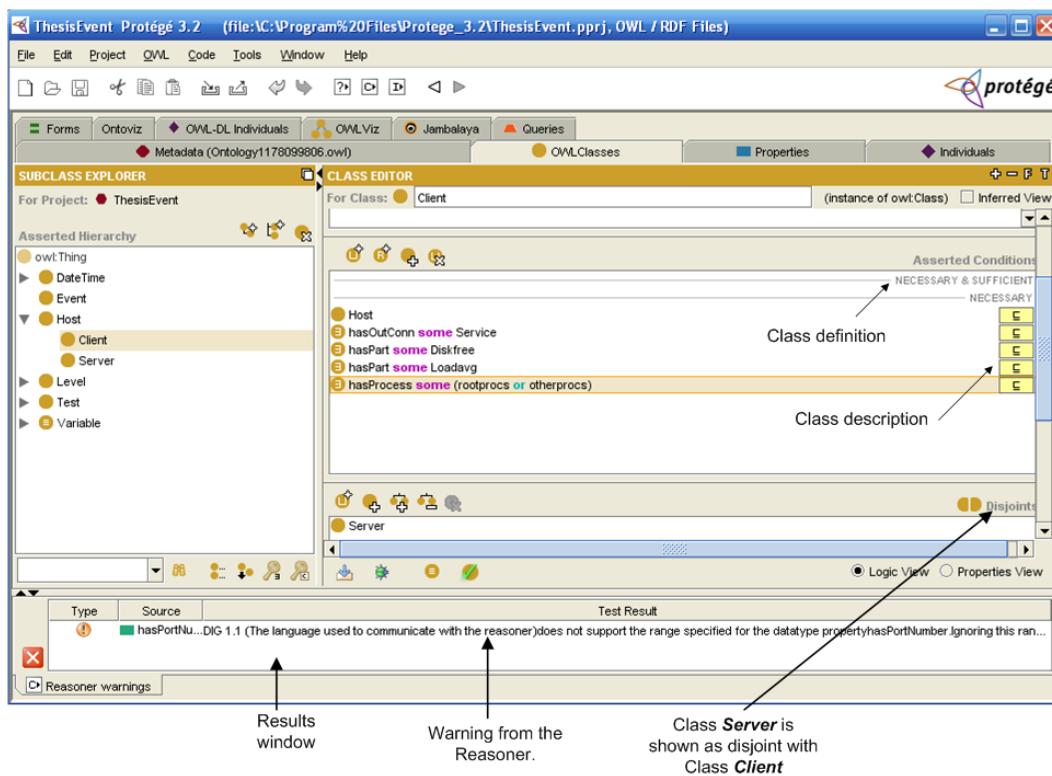

Figure 5.10: .
Consistency check of the ontology. Note the log from the Racer reasoner showing check timing details.

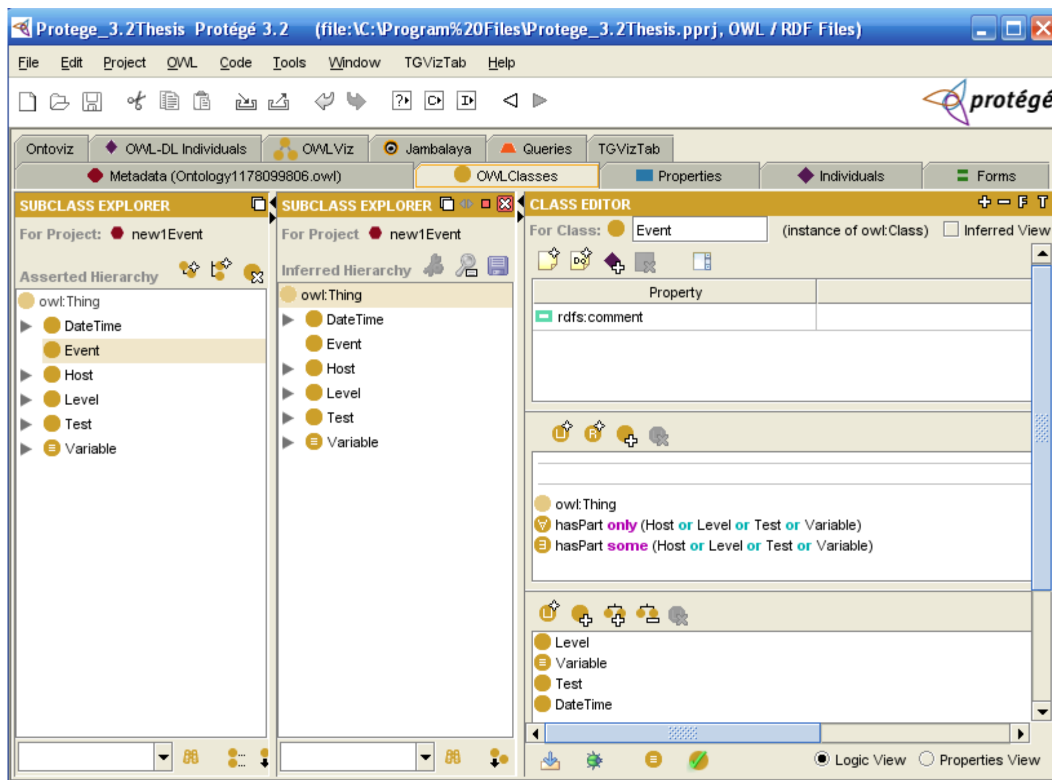

Figure 5.11: Classify taxonomy results to inferred hierarchy. This is done by the reasoner. Note the similarity of asserted and inferred hierarchies.

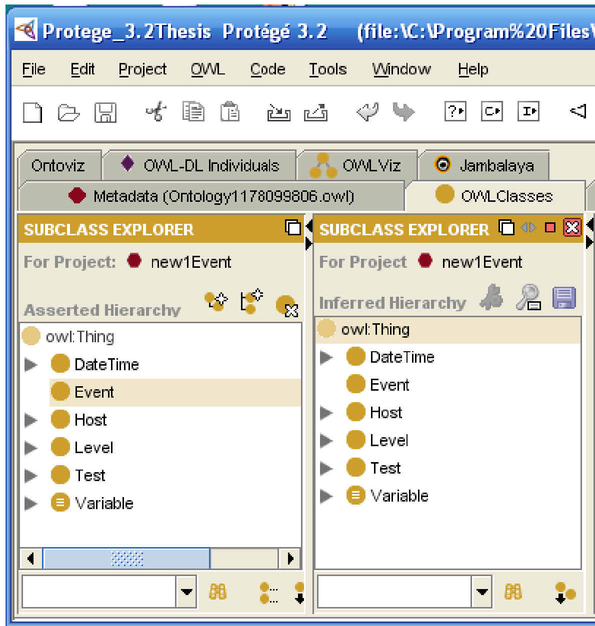

Figure 5.12: Classify taxonomy results to inferred hierarchy. This is done by the reasoner. Note the similarity of asserted and inferred hierarchies.

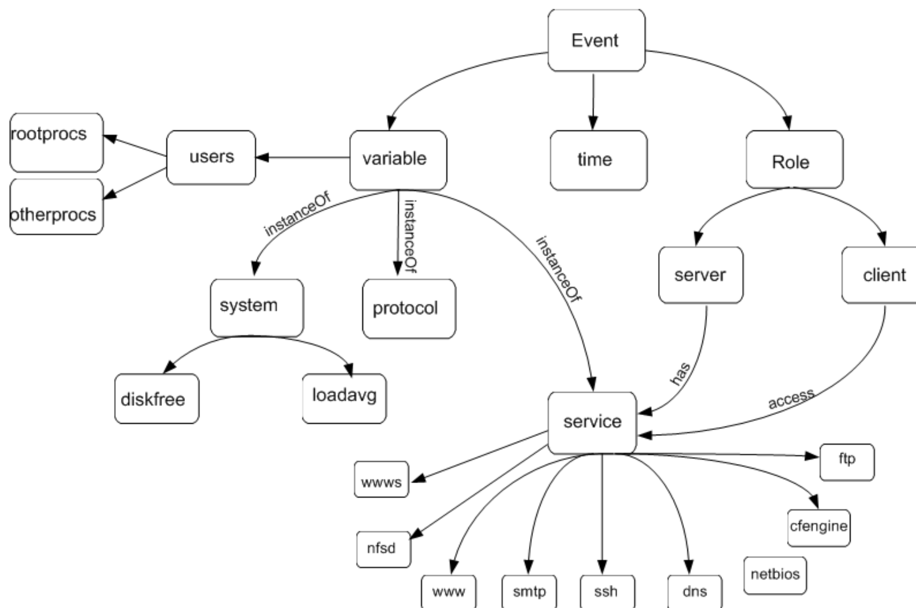

Figure 5.13: Cfengine events class hierarchy.

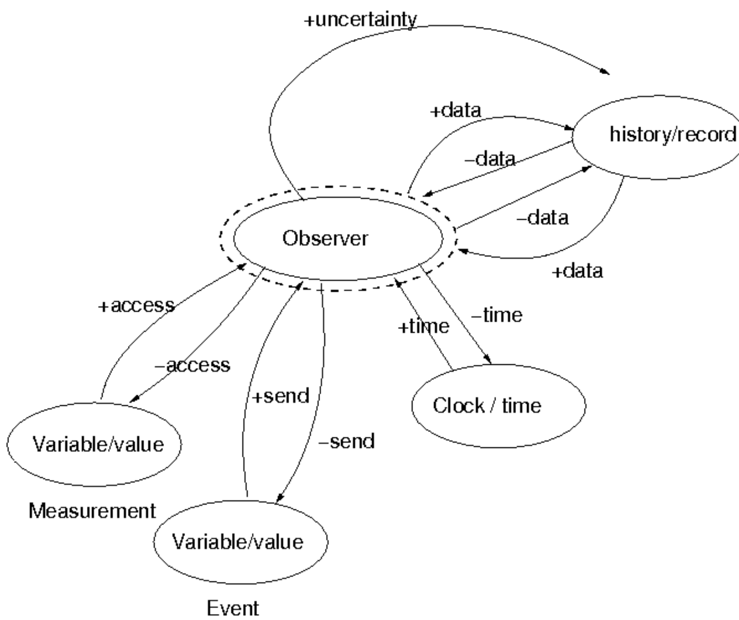

Figure 5.14: Promise graph for an observatory or observer (a behavioural instance of the concept of "observation"). It consists only of functional elements that promise something to one another. The notion of measurement is an emergent consequence of the fact that an agent promises to make its value available to an observer, and that the observer promises to use it. An event is represented as the promise to report on changes and the promise to listen to such reports. In promise theory, one sees the direction of the causality.

Chapter 6

Conclusions

In the course of this work we have examined anomaly detection using Cfengine, as a case study. In parallel, we examined ontology as a means for supporting filtering the multitude of events collected by the Anomaly Detection system of the case study. In this short thesis we cannot cover every possible aspect of ontologies for anomaly detection. Even though we mainly focused on Cfengine and its devloped ontology for anomaly detection, the scope is still too rich for a complete analysis in the time available.

A number of interesting questions came up during the ontology development phase, such as:

- How to decide whether to *describe* (using necessary condition) or *define* (using necessary and sufficient conditions) a class?. The available literature does not clarify this issue, and the decision is left for the ontology designer's intuition.
- Does the reasoner use the class *definitions* or *descriptions* or both to classify and perform consistency checking? Some literature indicates that the Reasoner does not use necessary conditions (Primitive classes) to classify the ontology. We found conflicting results when testing this fact and no plausible explanation.
- What is the significance of the hierarchical class representation in ontology development? Is there an alternative way of structuring ontologies without using hierarchies?

We think that, the question of whether we should put concepts in a hierarchy and how, was significant because from our experience we found that the logical classification of classes using class expressions requires not only the explicit intentional decision of ontology designer but also taking into account the inference mechanism of the reasoner. This is because, one can easily see and manipulate the hierarchical relations between classes (e.g by manually assertion of class hierarchy) but the logical hierarchical relations inferred by the reasoner exists and needs to be considered in design decisions too.

We observed that the use of asserting hierarchical relations with Protege OWL Plugin simplifies extension of subclasses by only modifying the superclass, the changes are inherited down the hierarchy. That is, asserting a certain property or restriction on

a superclass is the same as asserting that property or restriction to all individuals in the subclasses of the superclass. This observation agrees with the previous one about the need for careful classification of classes to avoid inconsistencies.

We think that, effective representation of knowledge for the purpose of solving a certain problem or doing a certain task, strongly depends on the nature of the problem or task and the strategy for inferencing to be applied to the problem. Since OWL as an ontology representation language was designed specifically for the development of the Semantic Web and mostly for knowledge sharing, is it appropriate and effective for representing knowledge for different purposes other than knowledge sharing?

We found that, ontology in its nature is somewhat analogous to Anomaly Detection paradigm in the sense that, ontology provides a framework for specifying and describing concepts and their relations in a domain of interest. This corresponds to describing the ″normal or expected state (behaviour)″ of entities in a domain. This is analogous to an Anomaly Detection system which through learning process establishes a 'normal' state of a host/network and use this baseline 'knowledge' to identify or recognize significant changes. The similarity is seen when one considers that an ontology describes the current/ existing knowledge about the domain and has provisions for extension of this knowledge, just as an Anomaly Detection system knowing normal state, can identify unexpected/unknown behaviour of a system. This is a kind of an Open World Assumption.

since an ontology does not (and probably will not) provide a complete set of terms in a domain, what would be important is to be systematic and to be able to ensure openness and extendibility in the approach.

In addition to ontology, we looked at some other knowledge representation formalisms namely, UML, Topic Maps and Promise Theory. The usability of UML and Topic Maps for ontology representation is being researched by a number of communities, and from literature, we found no striking similarity or insights on how they can be used to develop a conceptual model for our ontology. However, we found some similarities and differences between ontology and Promise Theory in terms of knowledge representation. We think that Promise Theory warrants further research in relation to ontology, specifically, Task ontology drawing existing experience from ontologies developed for SOA applications.

We think that through the use of axioms and rules, ontology can define completely the values that a concept can have, thus describe certain behaviours. Since with anomaly detection we are concerned with behaviours of a system and entities affecting the state / behaviour of a system, we think that ontology has the potential for supporting filtering of anomaly events through appropriate representation formalism and inference strategy.

6.1 Future Work

Further research on the suitability of OWL for effective representation of knowledge intended for different purposes other than knowledge sharing is recommended. This might shed more light on how best can ontology be used to accomplish a task of supporting event filtering. A comparison can then be made between the effectiveness

of a domain ontology versus task ontology for filtering anomaly events. In addition, the techniques such as Problem-solving can be integrated in the Task ontology and tested.

Research on how Promise Theory can be used to represent knowledge about and within events or entities in an Anomaly Detection system as an input to an ontology, to explore how effective that can be in filtering, clustering or correlating events in the efforts to identify interesting events.

Appendix A

Sample event

The following is an example of what we have been referring to as an*event, which is actually an anomaly event captured by the Servers* Nexus *and* Cube *in our case study network.*

```
    cf:nexus: LOW ENTROPY Incoming www anomaly high anomaly dev!! on nexus/Wed
cf:nexus: ------------------------------------------------------------------------
cf:nexus: In the last 40 minutes, the peak state was q = 129:
 {
 DNS key: 192.193.245.15 = 192.193.245.15 (1/129)
 DNS key: 83.143.10.190 = 10-190.vgccl.net (1/129)
 DNS key: 57.66.152.190 = 57.66.152.190 (108/129)
 DNS key: 202.156.11.4 = 202-156-11-4.cache.maxonline.com.sg (8/129)
 DNS key: 221.135.230.58 = 221-135-230-58.sify.net (3/129)
 DNS key: 59.93.86.21 = 59.93.86.21 (2/129)
 DNS key: 80.239.38.65 = flamme.gulesider.no (1/129)
 DNS key: 74.6.71.45 = lj611562.inktomisearch.com (1/129)
 DNS key: 83.255.186.222 = c83-255-186-222.bredband.comhem.se (2/129)
 DNS key: 74.6.74.214 = lj612588.inktomisearch.com (1/129)
 DNS key: 81.93.168.213 = trk-wiki01.osl.basefarm.net (1/129)
 -
 Frequency: 57.66.152.190    |**********************************************
 Frequency: 202.156.11.4     |********        (8/129)
 Frequency: 221.135.230.58   |***         (3/129)
 Frequency: 83.255.186.222   |**         (2/129)
 Frequency: 59.93.86.21      |**         (2/129)
 Frequency: 81.93.168.213    |*         (1/129)
 Frequency: 74.6.74.214      |*         (1/129)
 Frequency: 74.6.71.45       |*         (1/129)
 Frequency: 80.239.38.65     |*         (1/129)
 Frequency: 83.143.10.190    |*         (1/129)
 Frequency: 192.193.245.15   |*         (1/129)
 }
 -
```

```
 Scaled entropy of addresses = 0.8 %
 (Entropy = 0 for single source, 100 for flatly distributed source)
 -
cf:nexus: ------------------------------------------------------------
cf:nexus: State of incoming.www peaked at Wed Mar 28 18:13:59 2007

cf:cube: LOW ENTROPY Incoming www anomaly high anomaly dev!! on cube/Sat May
cf:cube: ------------------------------------------------------------
cf:cube: In the last 40 minutes, the peak state was q = 86:
 {
 DNS key: 220.168.124.84 = 220.168.124.84 (78/86)
 DNS key: 81.191.146.160 = cA092BF51.dhcp.bluecom.no (1/86)
 DNS key: 84.209.15.231 = cm-84.209.15.231.chello.no (2/86)
 DNS key: 74.6.71.243 = lj611467.inktomisearch.com (1/86)
 DNS key: 74.6.66.55 = lj612258.inktomisearch.com (1/86)
 DNS key: 74.6.65.248 = lj612276.inktomisearch.com (1/86)
 DNS key: 72.197.90.197 = ip72-197-90-197.sd.sd.cox.net (1/86)
 DNS key: 65.54.188.57 = livebot-65-54-188-57.search.live.com (1/86)
 -
 Frequency: 220.168.124.84   |*********************************************
 Frequency: 84.209.15.231    |**         (2/86)
 Frequency: 65.54.188.57     |*         (1/86)
 Frequency: 72.197.90.197    |*         (1/86)
 Frequency: 74.6.65.248      |*         (1/86)
 Frequency: 74.6.66.55       |*         (1/86)
 Frequency: 74.6.71.243      |*         (1/86)
 Frequency: 81.191.146.160   |*         (1/86)
 }
 -
 Scaled entropy of addresses = 10.9 %
 (Entropy = 0 for single source, 100 for flatly distributed source)
 -
cf:cube: ------------------------------------------------------------
cf:cube: State of incoming.www peaked at Sat May  5 11:30:28 2007
```

Both anomaly events shown are of typeentropy_www_in_high. As mentioned previously, due to the fact that too many events have their numerical values exceeding thresholds determined by an arbitrary policy, there was a need to identify other criteria to pin down which anomalies are interesting and which are not. Therefore, as a second level of policy filtering, Cfengine provides a measure of the ***entropy*** *of the source IP addresses of the measured data. A low entropy value (as in this case) means that most of the events came from only a few (or one) IP addresses. This extra information might help in deciding whether an event is interesting or not. Note the Scaled entropy of addresses = 0.8% and 10.9% for Nexus and Cube respectively.*

Appendix B

Sample event log

```
2006-09-14-12-40 entropy_smtp_in_low
2006-09-14-12-40 entropy_tcpsyn_in_low
2006-09-14-12-40 entropy_dns_in_low
2006-09-14-12-40 otherprocs_high_ldt
2006-09-14-12-40 www_in_high_ldt
2006-09-14-12-40 loadavg_high_ldt
2006-09-14-12-40 users_high_ldt
2006-09-14-13-10 entropy_smtp_in_low
2006-09-14-13-10 entropy_tcpsyn_in_low
2006-09-14-13-10 entropy_dns_in_low
2006-09-14-13-10 ssh_in_high_ldt
2006-09-14-13-10 users_high_ldt
2006-09-14-13-40 www_in_high_dev2
2006-09-14-13-40 entropy_smtp_in_low
2006-09-14-13-40 entropy_tcpsyn_in_low
2006-09-14-13-40 entropy_dns_in_low
2006-09-14-13-40 ssh_in_high_ldt
2006-09-14-13-40 rootprocs_high_ldt
2006-09-14-13-40 otherprocs_high_ldt
2006-09-14-13-40 www_in_high_ldt
2006-09-14-13-40 smtp_out_high_ldt
2006-09-14-13-40 loadavg_high_ldt
2006-09-14-14-10 entropy_smtp_in_low
2006-09-14-14-10 entropy_tcpsyn_in_low
2006-09-14-14-10 entropy_dns_in_low
2006-09-14-14-40 entropy_smtp_in_low
2006-09-14-14-40 entropy_tcpsyn_in_low
2006-09-14-14-40 entropy_dns_in_low
2006-09-14-14-40 otherprocs_high_ldt
2006-09-14-15-10 entropy_www_in_high
2006-09-14-15-10 entropy_smtp_in_low
2006-09-14-15-10 entropy_tcpsyn_in_low
```

```
2006-09-14-15-10 entropy_dns_in_low
2006-09-14-15-10 loadavg_high_ldt
```

The given excerpt is part of event log created and updated by cfagent every 30 minutes. Events collected since last time cfagent run are shown by the same timestamp. A long list of events like this is what a system administrator has to analyse and identify interesting events. We have developed an ontology to support filtering of such events.

Appendix C

List of Cfengine events

As mentioned previously, when cfengine detects an anomaly, it classified the current statistical state of the system into a number of classes. Below is the list of all possible anomaly events that currently can be detected by Cfengine anomaly detection system.

```
users_high_microanomaly
users_low_microanomaly
users_high_dev1
users_low_dev1
users_high_dev2
users_low_dev2
users_high_anomaly
users_low_anomaly
users_high_ldt
users_low_ldt
rootprocs_high_microanomaly
rootprocs_low_microanomaly
rootprocs_high_dev1
rootprocs_low_dev1
rootprocs_high_dev2
rootprocs_low_dev2
rootprocs_high_anomaly
rootprocs_low_anomaly
rootprocs_high_ldt
rootprocs_low_ldt
otherprocs_high_microanomaly
otherprocs_low_microanomaly
otherprocs_high_dev1
otherprocs_low_dev1
otherprocs_high_dev2
otherprocs_low_dev2
otherprocs_high_anomaly
otherprocs_low_anomaly
```

```
otherprocs_high_ldt
otherprocs_low_ldt
diskfree_high_microanomaly
diskfree_low_microanomaly
diskfree_high_dev1
diskfree_low_dev1
diskfree_high_dev2
diskfree_low_dev2
diskfree_high_anomaly
diskfree_low_anomaly
diskfree_high_ldt
diskfree_low_ldt
loadavg_high_microanomaly
loadavg_low_microanomaly
loadavg_high_dev1
loadavg_low_dev1
loadavg_high_dev2
loadavg_low_dev2
loadavg_high_anomaly
loadavg_low_anomaly
loadavg_high_ldt
loadavg_low_ldt
netbiosns_in_high_microanomaly
netbiosns_in_low_microanomaly
netbiosns_in_high_dev1
netbiosns_in_low_dev1
netbiosns_in_high_dev2
netbiosns_in_low_dev2
netbiosns_in_high_anomaly
netbiosns_in_low_anomaly
netbiosns_in_high_ldt
netbiosns_in_low_ldt
netbiosns_out_high_microanomaly
netbiosns_out_low_microanomaly
netbiosns_out_high_dev1
netbiosns_out_low_dev1
netbiosns_out_high_dev2
netbiosns_out_low_dev2
netbiosns_out_high_anomaly
netbiosns_out_low_anomaly
netbiosns_out_high_ldt
netbiosns_out_low_ldt
netbiosdgm_in_high_microanomaly
netbiosdgm_in_low_microanomaly
netbiosdgm_in_high_dev1
```

```
netbiosdgm_in_low_dev1
netbiosdgm_in_high_dev2
netbiosdgm_in_low_dev2
netbiosdgm_in_high_anomaly
netbiosdgm_in_low_anomaly
netbiosdgm_in_high_ldt
netbiosdgm_in_low_ldt
netbiosdgm_out_high_microanomaly
netbiosdgm_out_low_microanomaly
netbiosdgm_out_high_dev1
netbiosdgm_out_low_dev1
netbiosdgm_out_high_dev2
netbiosdgm_out_low_dev2
netbiosdgm_out_high_anomaly
netbiosdgm_out_low_anomaly
netbiosdgm_out_high_ldt
netbiosdgm_out_low_ldt
netbiosssn_in_high_microanomaly
netbiosssn_in_low_microanomaly
netbiosssn_in_high_dev1
netbiosssn_in_low_dev1
netbiosssn_in_high_dev2
netbiosssn_in_low_dev2
netbiosssn_in_high_anomaly
netbiosssn_in_low_anomaly
netbiosssn_in_high_ldt
netbiosssn_in_low_ldt
netbiosssn_out_high_microanomaly
netbiosssn_out_low_microanomaly
netbiosssn_out_high_dev1
netbiosssn_out_low_dev1
netbiosssn_out_high_dev2
netbiosssn_out_low_dev2
netbiosssn_out_high_anomaly
netbiosssn_out_low_anomaly
netbiosssn_out_high_ldt
netbiosssn_out_low_ldt
irc_in_high_microanomaly
irc_in_low_microanomaly
irc_in_high_dev1
irc_in_low_dev1
irc_in_high_dev2
irc_in_low_dev2
irc_in_high_anomaly
irc_in_low_anomaly
```

```
irc_in_high_ldt
irc_in_low_ldt
irc_out_high_microanomaly
irc_out_low_microanomaly
irc_out_high_dev1
irc_out_low_dev1
irc_out_high_dev2
irc_out_low_dev2
irc_out_high_anomaly
irc_out_low_anomaly
irc_out_high_ldt
irc_out_low_ldt
cfengine_in_high_microanomaly
cfengine_in_low_microanomaly
cfengine_in_high_dev1
cfengine_in_low_dev1
cfengine_in_high_dev2
cfengine_in_low_dev2
cfengine_in_high_anomaly
cfengine_in_low_anomaly
cfengine_in_high_ldt
cfengine_in_low_ldt
cfengine_out_high_microanomaly
cfengine_out_low_microanomaly
cfengine_out_high_dev1
cfengine_out_low_dev1
cfengine_out_high_dev2
cfengine_out_low_dev2
cfengine_out_high_anomaly
cfengine_out_low_anomaly
cfengine_out_high_ldt
cfengine_out_low_ldt
nfsd_in_high_microanomaly
nfsd_in_low_microanomaly
nfsd_in_high_dev1
nfsd_in_low_dev1
nfsd_in_high_dev2
nfsd_in_low_dev2
nfsd_in_high_anomaly
nfsd_in_low_anomaly
nfsd_in_high_ldt
nfsd_in_low_ldt
nfsd_out_high_microanomaly
nfsd_out_low_microanomaly
nfsd_out_high_dev1
```

```
nfsd_out_low_dev1
nfsd_out_high_dev2
nfsd_out_low_dev2
nfsd_out_high_anomaly
nfsd_out_low_anomaly
nfsd_out_high_ldt
nfsd_out_low_ldt
smtp_in_high_microanomaly
smtp_in_low_microanomaly
smtp_in_high_dev1
smtp_in_low_dev1
smtp_in_high_dev2
smtp_in_low_dev2
smtp_in_high_anomaly
smtp_in_low_anomaly
smtp_in_high_ldt
smtp_in_low_ldt
smtp_out_high_microanomaly
smtp_out_low_microanomaly
smtp_out_high_dev1
smtp_out_low_dev1
smtp_out_high_dev2
smtp_out_low_dev2
smtp_out_high_anomaly
smtp_out_low_anomaly
smtp_out_high_ldt
smtp_out_low_ldt
www_in_high_microanomaly
www_in_low_microanomaly
www_in_high_dev1
www_in_low_dev1
www_in_high_dev2
www_in_low_dev2
www_in_high_anomaly
www_in_low_anomaly
www_in_high_ldt
www_in_low_ldt
www_out_high_microanomaly
www_out_low_microanomaly
www_out_high_dev1
www_out_low_dev1
www_out_high_dev2
www_out_low_dev2
www_out_high_anomaly
www_out_low_anomaly
```

```
www_out_high_ldt
www_out_low_ldt
ftp_in_high_microanomaly
ftp_in_low_microanomaly
ftp_in_high_dev1
ftp_in_low_dev1
ftp_in_high_dev2
ftp_in_low_dev2
ftp_in_high_anomaly
ftp_in_low_anomaly
ftp_in_high_ldt
ftp_in_low_ldt
ftp_out_high_microanomaly
ftp_out_low_microanomaly
ftp_out_high_dev1
ftp_out_low_dev1
ftp_out_high_dev2
ftp_out_low_dev2
ftp_out_high_anomaly
ftp_out_low_anomaly
ftp_out_high_ldt
ftp_out_low_ldt
ssh_in_high_microanomaly
ssh_in_low_microanomaly
ssh_in_high_dev1
ssh_in_low_dev1
ssh_in_high_dev2
ssh_in_low_dev2
ssh_in_high_anomaly
ssh_in_low_anomaly
ssh_in_high_ldt
ssh_in_low_ldt
ssh_out_high_microanomaly
ssh_out_low_microanomaly
ssh_out_high_dev1
ssh_out_low_dev1
ssh_out_high_dev2
ssh_out_low_dev2
ssh_out_high_anomaly
ssh_out_low_anomaly
ssh_out_high_ldt
ssh_out_low_ldt
wwws_in_high_microanomaly
wwws_in_low_microanomaly
wwws_in_high_dev1
```

```
wwws_in_low_dev1
wwws_in_high_dev2
wwws_in_low_dev2
wwws_in_high_anomaly
wwws_in_low_anomaly
wwws_in_high_ldt
wwws_in_low_ldt
wwws_out_high_microanomaly
wwws_out_low_microanomaly
wwws_out_high_dev1
wwws_out_low_dev1
wwws_out_high_dev2
wwws_out_low_dev2
wwws_out_high_anomaly
wwws_out_low_anomaly
wwws_out_high_ldt
wwws_out_low_ldt
icmp_in_high_microanomaly
icmp_in_low_microanomaly
icmp_in_high_dev1
icmp_in_low_dev1
icmp_in_high_dev2
icmp_in_low_dev2
icmp_in_high_anomaly
icmp_in_low_anomaly
icmp_in_high_ldt
icmp_in_low_ldt
icmp_out_high_microanomaly
icmp_out_low_microanomaly
icmp_out_high_dev1
icmp_out_low_dev1
icmp_out_high_dev2
icmp_out_low_dev2
icmp_out_high_anomaly
icmp_out_low_anomaly
icmp_out_high_ldt
icmp_out_low_ldt
udp_in_high_microanomaly
udp_in_low_microanomaly
udp_in_high_dev1
udp_in_low_dev1
udp_in_high_dev2
udp_in_low_dev2
udp_in_high_anomaly
udp_in_low_anomaly
```

```
udp_in_high_ldt
udp_in_low_ldt
udp_out_high_microanomaly
udp_out_low_microanomaly
udp_out_high_dev1
udp_out_low_dev1
udp_out_high_dev2
udp_out_low_dev2
udp_out_high_anomaly
udp_out_low_anomaly
udp_out_high_ldt
udp_out_low_ldt
dns_in_high_microanomaly
dns_in_low_microanomaly
dns_in_high_dev1
dns_in_low_dev1
dns_in_high_dev2
dns_in_low_dev2
dns_in_high_anomaly
dns_in_low_anomaly
dns_in_high_ldt
dns_in_low_ldt
dns_out_high_microanomaly
dns_out_low_microanomaly
dns_out_high_dev1
dns_out_low_dev1
dns_out_high_dev2
dns_out_low_dev2
dns_out_high_anomaly
dns_out_low_anomaly
dns_out_high_ldt
dns_out_low_ldt
tcpsyn_in_high_microanomaly
tcpsyn_in_low_microanomaly
tcpsyn_in_high_dev1
tcpsyn_in_low_dev1
tcpsyn_in_high_dev2
tcpsyn_in_low_dev2
tcpsyn_in_high_anomaly
tcpsyn_in_low_anomaly
tcpsyn_in_high_ldt
tcpsyn_in_low_ldt
tcpsyn_out_high_microanomaly
tcpsyn_out_low_microanomaly
tcpsyn_out_high_dev1
```

```
tcpsyn_out_low_dev1
tcpsyn_out_high_dev2
tcpsyn_out_low_dev2
tcpsyn_out_high_anomaly
tcpsyn_out_low_anomaly
tcpsyn_out_high_ldt
tcpsyn_out_low_ldt
tcpack_in_high_microanomaly
tcpack_in_low_microanomaly
tcpack_in_high_dev1
tcpack_in_low_dev1
tcpack_in_high_dev2
tcpack_in_low_dev2
tcpack_in_high_anomaly
tcpack_in_low_anomaly
tcpack_in_high_ldt
tcpack_in_low_ldt
tcpack_out_high_microanomaly
tcpack_out_low_microanomaly
tcpack_out_high_dev1
tcpack_out_low_dev1
tcpack_out_high_dev2
tcpack_out_low_dev2
tcpack_out_high_anomaly
tcpack_out_low_anomaly
tcpack_out_high_ldt
tcpack_out_low_ldt
tcpfin_in_high_microanomaly
tcpfin_in_low_microanomaly
tcpfin_in_high_dev1
tcpfin_in_low_dev1
tcpfin_in_high_dev2
tcpfin_in_low_dev2
tcpfin_in_high_anomaly
tcpfin_in_low_anomaly
tcpfin_in_high_ldt
tcpfin_in_low_ldt
tcpfin_out_high_microanomaly
tcpfin_out_low_microanomaly
tcpfin_out_high_dev1
tcpfin_out_low_dev1
tcpfin_out_high_dev2
tcpfin_out_low_dev2
tcpfin_out_high_anomaly
tcpfin_out_low_anomaly
```

```
tcpfin_out_high_ldt
tcpfin_out_low_ldt
tcpmisc_in_high_microanomaly
tcpmisc_in_low_microanomaly
tcpmisc_in_high_dev1
tcpmisc_in_low_dev1
tcpmisc_in_high_dev2
tcpmisc_in_low_dev2
tcpmisc_in_high_anomaly
tcpmisc_in_low_anomaly
tcpmisc_in_high_ldt
tcpmisc_in_low_ldt
tcpmisc_out_high_microanomaly
tcpmisc_out_low_microanomaly
tcpmisc_out_high_dev1
tcpmisc_out_low_dev1
tcpmisc_out_high_dev2
tcpmisc_out_low_dev2
tcpmisc_out_high_anomaly
tcpmisc_out_low_anomaly
tcpmisc_out_high_ldt
tcpmisc_out_low_ldt
```

Appendix D

Graphical view of Results from Protege

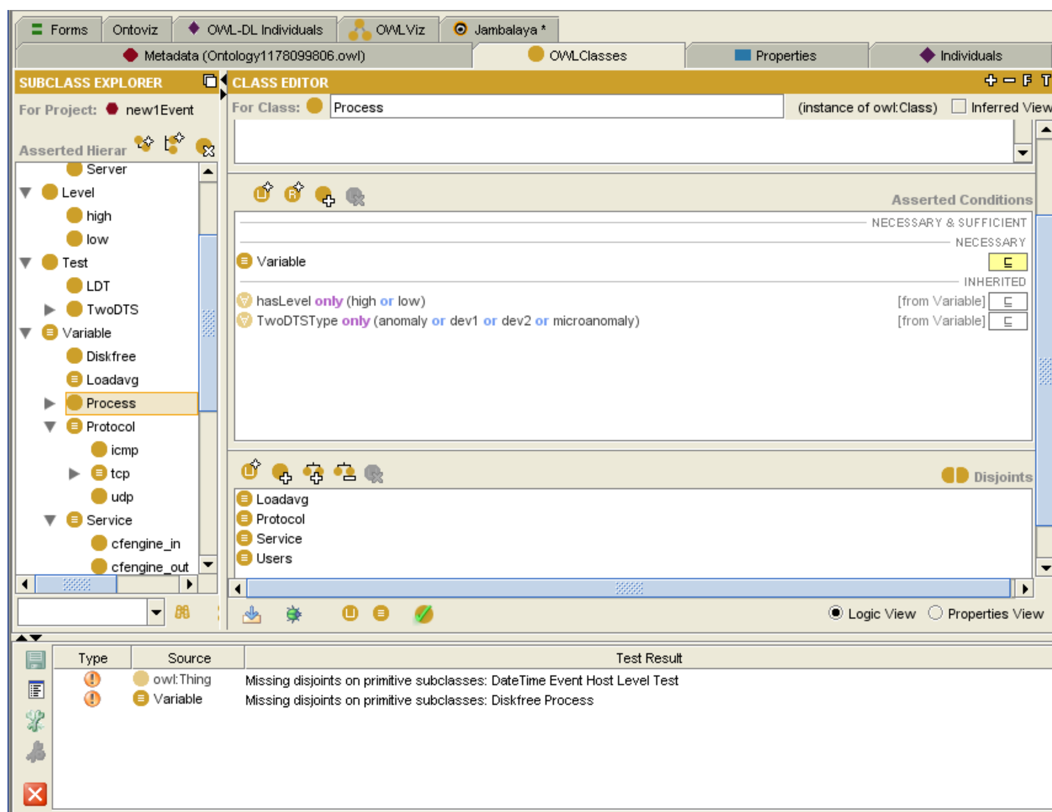

Figure D.1: Consistency check shows whether there is any inconsistent class. This is done by the reasoner.

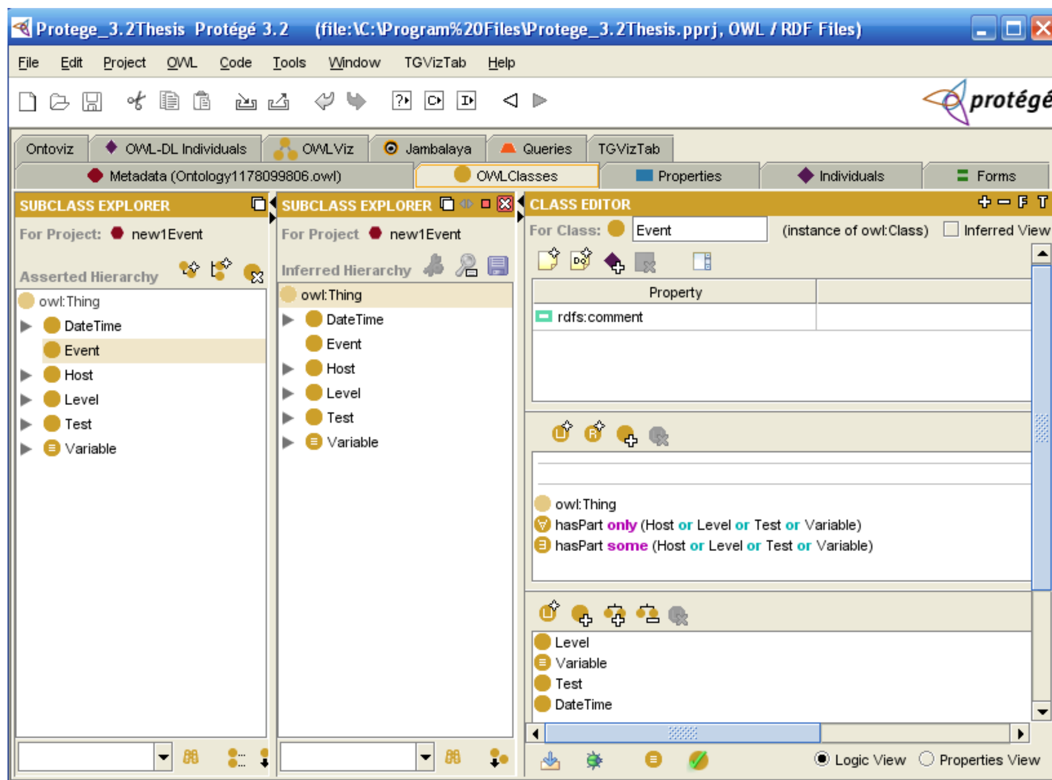

Figure D.2: Classify taxonomy results to inferred hierarchy. This is done by the reasoner. Note the similarity of asserted and inferred hierarchies.

Appendix E

RDF/XML

```
    <?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/Ontology1178099806.owl#"
     xml:base="http://www.owl-ontologies.com/Ontology1178099806.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=""/>
    <owl:AllDifferent>
        <owl:distinctMembers rdf:parseType="Collection"/>
    </owl:AllDifferent>
    <owl:AllDifferent>
        <owl:distinctMembers rdf:parseType="Collection"/>
    </owl:AllDifferent>
    <owl:Class rdf:ID="anomaly">
        <rdfs:subClassOf rdf:resource="#TwoDTS"/>
        <owl:disjointWith rdf:resource="#dev2"/>
        <owl:disjointWith rdf:resource="#dev1"/>
        <owl:disjointWith rdf:resource="#microanomaly"/>
    </owl:Class>
```

```
<owl:Class rdf:ID="cfengine_in">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#isOwnedBy"/>
            <owl:someValuesFrom>
                <owl:Class>
                    <owl:unionOf rdf:parseType="Collection">
                        <owl:Class rdf:about="#Process"/>
                        <owl:Class rdf:about="#Users"/>
                    </owl:unionOf>
                </owl:Class>
            </owl:someValuesFrom>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="#Service"/>
    <owl:disjointWith rdf:resource="#dns_in"/>
    <owl:disjointWith rdf:resource="#ssh_in"/>
    <owl:disjointWith rdf:resource="#www_in"/>
    <owl:disjointWith rdf:resource="#cfengine_out"/>
    <owl:disjointWith rdf:resource="#nfsd_in"/>
    <owl:disjointWith rdf:resource="#smtp_out"/>
    <owl:disjointWith rdf:resource="#ssh_out"/>
    <owl:disjointWith rdf:resource="#dns_out"/>
    <owl:disjointWith rdf:resource="#nfsd_out"/>
    <owl:disjointWith rdf:resource="#ftp_out"/>
    <owl:disjointWith rdf:resource="#smtp_in"/>
    <owl:disjointWith rdf:resource="#ftp_in"/>
    <owl:disjointWith rdf:resource="#nsfd_out"/>
    <owl:disjointWith rdf:resource="#www_out"/>
</owl:Class>
<owl:Class rdf:ID="cfengine_out">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasInfluence"/>
            <owl:someValuesFrom rdf:resource="#rootprocs"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="#Service"/>
    <owl:disjointWith rdf:resource="#nsfd_out"/>
    <owl:disjointWith rdf:resource="#dns_out"/>
    <owl:disjointWith rdf:resource="#cfengine_in"/>
    <owl:disjointWith rdf:resource="#nfsd_in"/>
    <owl:disjointWith rdf:resource="#ssh_out"/>
    <owl:disjointWith rdf:resource="#smtp_out"/>
    <owl:disjointWith rdf:resource="#www_in"/>
```

```
    <owl:disjointWith rdf:resource="#smtp_in"/>
    <owl:disjointWith rdf:resource="#ftp_in"/>
    <owl:disjointWith rdf:resource="#dns_in"/>
    <owl:disjointWith rdf:resource="#www_out"/>
    <owl:disjointWith rdf:resource="#nfsd_out"/>
    <owl:disjointWith rdf:resource="#ssh_in"/>
    <owl:disjointWith rdf:resource="#ftp_out"/>
</owl:Class>
<owl:Class rdf:ID="Client">
    <owl:equivalentClass>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#hasOutConn"/>
                    <owl:someValuesFrom rdf:resource="#Service"/>
                </owl:Restriction>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#hasPart"/>
                    <owl:someValuesFrom rdf:resource="#Diskfree"/>
                </owl:Restriction>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#hasPart"/>
                    <owl:someValuesFrom rdf:resource="#Loadavg"/>
                </owl:Restriction>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#hasProcess"/>
                    <owl:someValuesFrom>
                        <owl:Class>
                            <owl:unionOf rdf:parseType="Collection">
                                <owl:Class rdf:about="#otherprocs"/>
                                <owl:Class rdf:about="#rootprocs"/>
                            </owl:unionOf>
                        </owl:Class>
                    </owl:someValuesFrom>
                </owl:Restriction>
            </owl:intersectionOf>
        </owl:Class>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="#Host"/>
    <owl:disjointWith rdf:resource="#Server"/>
</owl:Class>
<Server rdf:ID="Cube"/>
<owl:Class rdf:ID="DateTime">
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
    <rdfs:subClassOf>
```

```
        <owl:Restriction>
            <owl:onProperty rdf:resource="#isPartOf"/>
            <owl:someValuesFrom rdf:resource="#Event"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <owl:disjointWith rdf:resource="#Event"/>
    <owl:disjointWith rdf:resource="#Host"/>
    <owl:disjointWith rdf:resource="#Test"/>
    <owl:disjointWith rdf:resource="#Variable"/>
    <owl:disjointWith rdf:resource="#Level"/>
</owl:Class>
<owl:Class rdf:ID="DayOfWeek">
    <rdfs:subClassOf rdf:resource="#DateTime"/>
    <owl:disjointWith rdf:resource="#Week"/>
    <owl:disjointWith rdf:resource="#Year"/>
    <owl:disjointWith rdf:resource="#Month"/>
    <owl:disjointWith rdf:resource="#Minute"/>
    <owl:disjointWith rdf:resource="#Hour"/>
    <owl:disjointWith rdf:resource="#Second"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="dependsOn"/>
<owl:Class rdf:ID="dev1">
    <rdfs:subClassOf rdf:resource="#TwoDTS"/>
    <owl:disjointWith rdf:resource="#dev2"/>
    <owl:disjointWith rdf:resource="#anomaly"/>
    <owl:disjointWith rdf:resource="#microanomaly"/>
</owl:Class>
<owl:Class rdf:ID="dev2">
    <rdfs:subClassOf rdf:resource="#TwoDTS"/>
    <owl:disjointWith rdf:resource="#dev1"/>
    <owl:disjointWith rdf:resource="#anomaly"/>
    <owl:disjointWith rdf:resource="#microanomaly"/>
</owl:Class>
<owl:Class rdf:ID="Diskfree">
    <rdfs:subClassOf rdf:resource="#Variable"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#isInfluencedBy"/>
            <owl:someValuesFrom rdf:resource="#smtp_in"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <owl:disjointWith rdf:resource="#Users"/>
    <owl:disjointWith rdf:resource="#Service"/>
    <owl:disjointWith rdf:resource="#Protocol"/>
    <owl:disjointWith rdf:resource="#Process"/>
```

```
    <owl:disjointWith rdf:resource="#Loadavg"/>
</owl:Class>
<owl:Class rdf:ID="dns_in">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasProtocol"/>
            <owl:someValuesFrom rdf:resource="#udp"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="#Service"/>
    <owl:disjointWith rdf:resource="#www_in"/>
    <owl:disjointWith rdf:resource="#ftp_in"/>
    <owl:disjointWith rdf:resource="#nsfd_out"/>
    <owl:disjointWith rdf:resource="#ftp_out"/>
    <owl:disjointWith rdf:resource="#cfengine_in"/>
    <owl:disjointWith rdf:resource="#www_out"/>
    <owl:disjointWith rdf:resource="#ssh_out"/>
    <owl:disjointWith rdf:resource="#smtp_in"/>
    <owl:disjointWith rdf:resource="#smtp_out"/>
    <owl:disjointWith rdf:resource="#ssh_in"/>
    <owl:disjointWith rdf:resource="#nfsd_out"/>
    <owl:disjointWith rdf:resource="#nfsd_in"/>
    <owl:disjointWith rdf:resource="#dns_out"/>
    <owl:disjointWith rdf:resource="#cfengine_out"/>
</owl:Class>
<owl:Class rdf:ID="dns_out">
    <rdfs:subClassOf rdf:resource="#Service"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasProtocol"/>
            <owl:someValuesFrom rdf:resource="#udp"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <owl:disjointWith rdf:resource="#smtp_out"/>
    <owl:disjointWith rdf:resource="#smtp_in"/>
    <owl:disjointWith rdf:resource="#ftp_out"/>
    <owl:disjointWith rdf:resource="#nfsd_out"/>
    <owl:disjointWith rdf:resource="#cfengine_in"/>
    <owl:disjointWith rdf:resource="#ftp_in"/>
    <owl:disjointWith rdf:resource="#cfengine_out"/>
    <owl:disjointWith rdf:resource="#dns_in"/>
    <owl:disjointWith rdf:resource="#ssh_in"/>
    <owl:disjointWith rdf:resource="#www_in"/>
    <owl:disjointWith rdf:resource="#nfsd_in"/>
    <owl:disjointWith rdf:resource="#www_out"/>
```

```
    <owl:disjointWith rdf:resource="#ssh_out"/>
    <owl:disjointWith rdf:resource="#nsfd_out"/>
</owl:Class>
<owl:Class rdf:ID="Event">
    <owl:equivalentClass>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#hasPart"/>
                    <owl:allValuesFrom>
                        <owl:Class>
                            <owl:unionOf rdf:parseType="Collection">
                                <owl:Class rdf:about="#Host"/>
                                <owl:Class rdf:about="#Level"/>
                                <owl:Class rdf:about="#Test"/>
                                <owl:Class rdf:about="#Variable"/>
                            </owl:unionOf>
                        </owl:Class>
                    </owl:allValuesFrom>
                </owl:Restriction>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#hasPart"/>
                    <owl:someValuesFrom>
                        <owl:Class>
                            <owl:unionOf rdf:parseType="Collection">
                                <owl:Class rdf:about="#Host"/>
                                <owl:Class rdf:about="#Level"/>
                                <owl:Class rdf:about="#Test"/>
                                <owl:Class rdf:about="#Variable"/>
                            </owl:unionOf>
                        </owl:Class>
                    </owl:someValuesFrom>
                </owl:Restriction>
            </owl:intersectionOf>
        </owl:Class>
    </owl:equivalentClass>
    <owl:disjointWith rdf:resource="#DateTime"/>
    <owl:disjointWith rdf:resource="#Test"/>
    <owl:disjointWith rdf:resource="#Variable"/>
    <owl:disjointWith rdf:resource="#Level"/>
</owl:Class>
<owl:Class rdf:ID="Friday">
    <rdfs:subClassOf rdf:resource="#Weekdays"/>
    <owl:disjointWith rdf:resource="#Thursday"/>
    <owl:disjointWith rdf:resource="#Wednesday"/>
```

```
        <owl:disjointWith rdf:resource="#Tuesday"/>
        <owl:disjointWith rdf:resource="#Monday"/>
    </owl:Class>
    <owl:Class rdf:ID="ftp_in">
        <rdfs:subClassOf rdf:resource="#Service"/>
        <owl:disjointWith rdf:resource="#ssh_out"/>
        <owl:disjointWith rdf:resource="#www_out"/>
        <owl:disjointWith rdf:resource="#dns_in"/>
        <owl:disjointWith rdf:resource="#cfengine_out"/>
        <owl:disjointWith rdf:resource="#ssh_in"/>
        <owl:disjointWith rdf:resource="#dns_out"/>
        <owl:disjointWith rdf:resource="#nsfd_out"/>
        <owl:disjointWith rdf:resource="#nfsd_out"/>
        <owl:disjointWith rdf:resource="#nfsd_in"/>
        <owl:disjointWith rdf:resource="#ftp_out"/>
        <owl:disjointWith rdf:resource="#cfengine_in"/>
        <owl:disjointWith rdf:resource="#smtp_in"/>
        <owl:disjointWith rdf:resource="#www_in"/>
        <owl:disjointWith rdf:resource="#smtp_out"/>
    </owl:Class>
    <owl:Class rdf:ID="ftp_out">
        <rdfs:subClassOf rdf:resource="#Service"/>
        <owl:disjointWith rdf:resource="#smtp_out"/>
        <owl:disjointWith rdf:resource="#dns_out"/>
        <owl:disjointWith rdf:resource="#ssh_in"/>
        <owl:disjointWith rdf:resource="#nfsd_in"/>
        <owl:disjointWith rdf:resource="#ftp_in"/>
        <owl:disjointWith rdf:resource="#cfengine_out"/>
        <owl:disjointWith rdf:resource="#smtp_in"/>
        <owl:disjointWith rdf:resource="#www_out"/>
        <owl:disjointWith rdf:resource="#dns_in"/>
        <owl:disjointWith rdf:resource="#www_in"/>
        <owl:disjointWith rdf:resource="#ssh_out"/>
        <owl:disjointWith rdf:resource="#nfsd_out"/>
        <owl:disjointWith rdf:resource="#nsfd_out"/>
        <owl:disjointWith rdf:resource="#cfengine_in"/>
    </owl:Class>
    <owl:ObjectProperty rdf:ID="hasAnomalyLevel"/>
    <owl:ObjectProperty rdf:ID="hasDirection"/>
    <owl:ObjectProperty rdf:ID="hasFlag"/>
    <owl:ObjectProperty rdf:ID="hasInConn">
        <owl:inverseOf rdf:resource="#hasOutConn"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:ID="hasInfluence">
        <owl:inverseOf rdf:resource="#isInfluencedBy"/>
```

```
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasLevel">
    <owl:inverseOf rdf:resource="#isLevelOf"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasOutConn">
    <owl:inverseOf rdf:resource="#hasInConn"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasOwner">
    <owl:inverseOf rdf:resource="#isOwnedBy"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasPart">
    <owl:inverseOf rdf:resource="#isPartOf"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasPort">
    <owl:inverseOf rdf:resource="#isPortOf"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="hasPortNumber">
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="hasProcess"/>
<owl:ObjectProperty rdf:ID="hasProtocol"/>
<owl:ObjectProperty rdf:ID="hasService">
    <owl:inverseOf rdf:resource="#isServiceOf"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasTest"/>
<owl:ObjectProperty rdf:ID="hasVariable">
    <owl:inverseOf rdf:resource="#isVariableOf"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="high">
    <rdfs:subClassOf rdf:resource="#Level"/>
    <owl:disjointWith rdf:resource="#low"/>
</owl:Class>
<owl:Class rdf:ID="Host">
    <owl:disjointWith rdf:resource="#DateTime"/>
    <owl:disjointWith rdf:resource="#Level"/>
    <owl:disjointWith rdf:resource="#Test"/>
</owl:Class>
<owl:Class rdf:ID="Hour">
    <rdfs:subClassOf rdf:resource="#DateTime"/>
    <owl:disjointWith rdf:resource="#Week"/>
    <owl:disjointWith rdf:resource="#Year"/>
    <owl:disjointWith rdf:resource="#Month"/>
    <owl:disjointWith rdf:resource="#DayOfWeek"/>
    <owl:disjointWith rdf:resource="#Minute"/>
    <owl:disjointWith rdf:resource="#Second"/>
```

```
</owl:Class>
<owl:Class rdf:ID="icmp">
    <rdfs:subClassOf rdf:resource="#Protocol"/>
    <owl:disjointWith rdf:resource="#udp"/>
    <owl:disjointWith rdf:resource="#tcp"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="isInfluencedBy">
    <owl:inverseOf rdf:resource="#hasInfluence"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isLevelOf">
    <owl:inverseOf rdf:resource="#hasLevel"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isOwnedBy">
    <owl:inverseOf rdf:resource="#hasOwner"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isPartOf">
    <owl:inverseOf rdf:resource="#hasPart"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isPortOf">
    <owl:inverseOf rdf:resource="#hasPort"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isServiceOf">
    <owl:inverseOf rdf:resource="#hasService"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isVariableOf">
    <owl:inverseOf rdf:resource="#hasVariable"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="LDT">
    <rdfs:subClassOf rdf:resource="#Test"/>
    <owl:disjointWith rdf:resource="#TwoDTS"/>
</owl:Class>
<owl:Class rdf:ID="Level">
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#isPartOf"/>
            <owl:someValuesFrom rdf:resource="#Event"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <owl:disjointWith rdf:resource="#Event"/>
    <owl:disjointWith rdf:resource="#Test"/>
    <owl:disjointWith rdf:resource="#Variable"/>
    <owl:disjointWith rdf:resource="#DateTime"/>
    <owl:disjointWith rdf:resource="#Host"/>
</owl:Class>
```

```
<owl:Class rdf:ID="Loadavg">
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#isInfluencedBy"/>
            <owl:someValuesFrom>
                <owl:Class>
                    <owl:unionOf rdf:parseType="Collection">
                        <owl:Class rdf:about="#nfsd_in"/>
                        <owl:Class rdf:about="#otherprocs"/>
                        <owl:Class rdf:about="#rootprocs"/>
                    </owl:unionOf>
                </owl:Class>
            </owl:someValuesFrom>
        </owl:Restriction>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="#Variable"/>
    <owl:disjointWith rdf:resource="#Users"/>
    <owl:disjointWith rdf:resource="#Service"/>
    <owl:disjointWith rdf:resource="#Protocol"/>
    <owl:disjointWith rdf:resource="#Process"/>
    <owl:disjointWith rdf:resource="#Diskfree"/>
</owl:Class>
<owl:Class rdf:ID="low">
    <rdfs:subClassOf rdf:resource="#Level"/>
    <owl:disjointWith rdf:resource="#high"/>
</owl:Class>
<owl:Class rdf:ID="microanomaly">
    <rdfs:subClassOf rdf:resource="#TwoDTS"/>
    <owl:disjointWith rdf:resource="#dev2"/>
    <owl:disjointWith rdf:resource="#dev1"/>
    <owl:disjointWith rdf:resource="#anomaly"/>
</owl:Class>
<owl:Class rdf:ID="Minute">
    <rdfs:subClassOf rdf:resource="#DateTime"/>
    <owl:disjointWith rdf:resource="#Week"/>
    <owl:disjointWith rdf:resource="#Year"/>
    <owl:disjointWith rdf:resource="#Month"/>
    <owl:disjointWith rdf:resource="#DayOfWeek"/>
    <owl:disjointWith rdf:resource="#Hour"/>
    <owl:disjointWith rdf:resource="#Second"/>
</owl:Class>
<owl:Class rdf:ID="Monday">
    <rdfs:subClassOf rdf:resource="#Weekdays"/>
    <owl:disjointWith rdf:resource="#Friday"/>
    <owl:disjointWith rdf:resource="#Thursday"/>
```

```
        <owl:disjointWith rdf:resource="#Wednesday"/>
        <owl:disjointWith rdf:resource="#Tuesday"/>
    </owl:Class>
    <owl:Class rdf:ID="Month">
        <rdfs:subClassOf rdf:resource="#DateTime"/>
        <owl:disjointWith rdf:resource="#Week"/>
        <owl:disjointWith rdf:resource="#Year"/>
        <owl:disjointWith rdf:resource="#DayOfWeek"/>
        <owl:disjointWith rdf:resource="#Minute"/>
        <owl:disjointWith rdf:resource="#Hour"/>
        <owl:disjointWith rdf:resource="#Second"/>
    </owl:Class>
    <Server rdf:ID="Nexus"/>
    <owl:Class rdf:ID="nfsd_in">
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasInfluence"/>
                <owl:someValuesFrom rdf:resource="#Loadavg"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf rdf:resource="#Service"/>
        <owl:disjointWith rdf:resource="#www_in"/>
        <owl:disjointWith rdf:resource="#ftp_in"/>
        <owl:disjointWith rdf:resource="#cfengine_out"/>
        <owl:disjointWith rdf:resource="#ftp_out"/>
        <owl:disjointWith rdf:resource="#smtp_in"/>
        <owl:disjointWith rdf:resource="#dns_in"/>
        <owl:disjointWith rdf:resource="#www_out"/>
        <owl:disjointWith rdf:resource="#nsfd_out"/>
        <owl:disjointWith rdf:resource="#dns_out"/>
        <owl:disjointWith rdf:resource="#smtp_out"/>
        <owl:disjointWith rdf:resource="#ssh_in"/>
        <owl:disjointWith rdf:resource="#nfsd_out"/>
        <owl:disjointWith rdf:resource="#cfengine_in"/>
        <owl:disjointWith rdf:resource="#ssh_out"/>
    </owl:Class>
    <owl:Class rdf:ID="nfsd_out">
        <rdfs:subClassOf rdf:resource="#Service"/>
        <owl:disjointWith rdf:resource="#cfengine_in"/>
        <owl:disjointWith rdf:resource="#www_in"/>
        <owl:disjointWith rdf:resource="#nfsd_in"/>
        <owl:disjointWith rdf:resource="#smtp_in"/>
        <owl:disjointWith rdf:resource="#ftp_out"/>
        <owl:disjointWith rdf:resource="#dns_out"/>
        <owl:disjointWith rdf:resource="#cfengine_out"/>
```

```
        <owl:disjointWith rdf:resource="#nsfd_out"/>
        <owl:disjointWith rdf:resource="#smtp_out"/>
        <owl:disjointWith rdf:resource="#dns_in"/>
        <owl:disjointWith rdf:resource="#ftp_in"/>
        <owl:disjointWith rdf:resource="#ssh_out"/>
        <owl:disjointWith rdf:resource="#www_out"/>
        <owl:disjointWith rdf:resource="#ssh_in"/>
    </owl:Class>
    <owl:Class rdf:ID="nsfd_out">
        <rdfs:subClassOf rdf:resource="#Service"/>
        <owl:disjointWith rdf:resource="#cfengine_in"/>
        <owl:disjointWith rdf:resource="#nfsd_out"/>
        <owl:disjointWith rdf:resource="#nfsd_in"/>
        <owl:disjointWith rdf:resource="#ssh_out"/>
        <owl:disjointWith rdf:resource="#smtp_in"/>
        <owl:disjointWith rdf:resource="#cfengine_out"/>
        <owl:disjointWith rdf:resource="#ssh_in"/>
        <owl:disjointWith rdf:resource="#smtp_out"/>
        <owl:disjointWith rdf:resource="#www_in"/>
        <owl:disjointWith rdf:resource="#www_out"/>
        <owl:disjointWith rdf:resource="#ftp_out"/>
        <owl:disjointWith rdf:resource="#dns_in"/>
        <owl:disjointWith rdf:resource="#dns_out"/>
        <owl:disjointWith rdf:resource="#ftp_in"/>
    </owl:Class>
    <owl:Class rdf:ID="otherprocs">
        <owl:equivalentClass>
            <owl:Class>
                <owl:intersectionOf rdf:parseType="Collection">
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#hasInfluence"/>
                        <owl:someValuesFrom rdf:resource="#Loadavg"/>
                    </owl:Restriction>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#isInfluencedBy"/>
                        <owl:someValuesFrom>
                            <owl:Class>
                                <owl:unionOf rdf:parseType="Collection">
                                    <owl:Class rdf:about="#ftp_out"/>
                                    <owl:Class rdf:about="#nsfd_out"/>
                                    <owl:Class rdf:about="#smtp_out"/>
                                    <owl:Class rdf:about="#ssh_out"/>
                                    <owl:Class rdf:about="#www_out"/>
                                </owl:unionOf>
                            </owl:Class>
```

```
                    </owl:someValuesFrom>
                </owl:Restriction>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#isInfluencedBy"/>
                    <owl:someValuesFrom>
                        <owl:Class>
                            <owl:unionOf rdf:parseType="Collection">
                                <owl:Class rdf:about="#ftp_in"/>
                                <owl:Class rdf:about="#smtp_in"/>
                                <owl:Class rdf:about="#ssh_in"/>
                                <owl:Class rdf:about="#Users"/>
                                <owl:Class rdf:about="#www_in"/>
                            </owl:unionOf>
                        </owl:Class>
                    </owl:someValuesFrom>
                </owl:Restriction>
            </owl:intersectionOf>
        </owl:Class>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="#Process"/>
    <owl:disjointWith rdf:resource="#rootprocs"/>
</owl:Class>
<owl:Class rdf:ID="Process">
    <rdfs:subClassOf rdf:resource="#Variable"/>
    <owl:disjointWith rdf:resource="#Users"/>
    <owl:disjointWith rdf:resource="#Service"/>
    <owl:disjointWith rdf:resource="#Protocol"/>
    <owl:disjointWith rdf:resource="#Loadavg"/>
    <owl:disjointWith rdf:resource="#Diskfree"/>
</owl:Class>
<owl:Class rdf:ID="Protocol">
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasPart"/>
            <owl:allValuesFrom>
                <owl:Class>
                    <owl:unionOf rdf:parseType="Collection">
                        <owl:Class rdf:about="#icmp"/>
                        <owl:Class rdf:about="#tcp"/>
                        <owl:Class rdf:about="#udp"/>
                    </owl:unionOf>
                </owl:Class>
            </owl:allValuesFrom>
        </owl:Restriction>
    </owl:equivalentClass>
```

```
    <rdfs:subClassOf rdf:resource="#Variable"/>
    <owl:disjointWith rdf:resource="#Users"/>
    <owl:disjointWith rdf:resource="#Service"/>
    <owl:disjointWith rdf:resource="#Process"/>
    <owl:disjointWith rdf:resource="#Loadavg"/>
    <owl:disjointWith rdf:resource="#Diskfree"/>
</owl:Class>
<Client rdf:ID="Rex"/>
<owl:Class rdf:ID="rootprocs">
    <owl:equivalentClass>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#hasInfluence"/>
                    <owl:someValuesFrom rdf:resource="#Loadavg"/>
                </owl:Restriction>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#isInfluencedBy"/>
                    <owl:someValuesFrom>
                        <owl:Class>
                            <owl:unionOf rdf:parseType="Collection">
                                <owl:Class rdf:about="#cfengine_out"/>
                                <owl:Class rdf:about="#smtp_in"/>
                                <owl:Class rdf:about="#ssh_in"/>
                            </owl:unionOf>
                        </owl:Class>
                    </owl:someValuesFrom>
                </owl:Restriction>
            </owl:intersectionOf>
        </owl:Class>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="#Process"/>
    <owl:disjointWith rdf:resource="#otherprocs"/>
</owl:Class>
<owl:Class rdf:ID="Saturday">
    <rdfs:subClassOf rdf:resource="#Weekends"/>
    <owl:disjointWith rdf:resource="#Sunday"/>
</owl:Class>
<Client rdf:ID="Satyagraha"/>
<owl:Class rdf:ID="Second">
    <rdfs:subClassOf rdf:resource="#DateTime"/>
    <owl:disjointWith rdf:resource="#Week"/>
    <owl:disjointWith rdf:resource="#Year"/>
    <owl:disjointWith rdf:resource="#Month"/>
    <owl:disjointWith rdf:resource="#DayOfWeek"/>
```

```
    <owl:disjointWith rdf:resource="#Minute"/>
    <owl:disjointWith rdf:resource="#Hour"/>
</owl:Class>
<owl:Class rdf:ID="Server">
    <rdfs:subClassOf rdf:resource="#Host"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasPart"/>
            <owl:someValuesFrom rdf:resource="#Loadavg"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasPart"/>
            <owl:someValuesFrom rdf:resource="#Diskfree"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasProcess"/>
            <owl:someValuesFrom>
                <owl:Class>
                    <owl:unionOf rdf:parseType="Collection">
                        <owl:Class rdf:about="#otherprocs"/>
                        <owl:Class rdf:about="#rootprocs"/>
                    </owl:unionOf>
                </owl:Class>
            </owl:someValuesFrom>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasInConn"/>
            <owl:someValuesFrom rdf:resource="#Service"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <owl:disjointWith rdf:resource="#Client"/>
</owl:Class>
<owl:Class rdf:ID="Service">
    <owl:equivalentClass>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#hasAnomalyLevel"/>
                    <owl:someValuesFrom>
```

```
                        <owl:Class>
                            <owl:unionOf rdf:parseType="Collection">
                                <owl:Class rdf:about="#anomaly"/>
                                <owl:Class rdf:about="#dev1"/>
                                <owl:Class rdf:about="#dev2"/>
                                <owl:Class rdf:about="#microanomaly"/>
                            </owl:unionOf>
                        </owl:Class>
                    </owl:someValuesFrom>
                </owl:Restriction>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#hasFlag"/>
                    <owl:someValuesFrom rdf:resource="#tcpFlags"/>
                </owl:Restriction>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#hasProcess"/>
                    <owl:someValuesFrom>
                        <owl:Class>
                            <owl:unionOf rdf:parseType="Collection">
                                <owl:Class rdf:about="#otherprocs"/>
                                <owl:Class rdf:about="#rootprocs"/>
                            </owl:unionOf>
                        </owl:Class>
                    </owl:someValuesFrom>
                </owl:Restriction>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#isServiceOf"/>
                    <owl:someValuesFrom rdf:resource="#Users"/>
                </owl:Restriction>
            </owl:intersectionOf>
        </owl:Class>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="#Variable"/>
    <owl:disjointWith rdf:resource="#Loadavg"/>
    <owl:disjointWith rdf:resource="#Users"/>
    <owl:disjointWith rdf:resource="#Process"/>
    <owl:disjointWith rdf:resource="#Diskfree"/>
    <owl:disjointWith rdf:resource="#Protocol"/>
</owl:Class>
<owl:Class rdf:ID="smtp_in">
    <rdfs:subClassOf rdf:resource="#Service"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasInfluence"/>
            <owl:someValuesFrom rdf:resource="#rootprocs"/>
```

```
        </owl:Restriction>
    </rdfs:subClassOf>
    <owl:disjointWith rdf:resource="#cfengine_in"/>
    <owl:disjointWith rdf:resource="#www_out"/>
    <owl:disjointWith rdf:resource="#ssh_in"/>
    <owl:disjointWith rdf:resource="#cfengine_out"/>
    <owl:disjointWith rdf:resource="#ftp_in"/>
    <owl:disjointWith rdf:resource="#nsfd_out"/>
    <owl:disjointWith rdf:resource="#nfsd_in"/>
    <owl:disjointWith rdf:resource="#ssh_out"/>
    <owl:disjointWith rdf:resource="#ftp_out"/>
    <owl:disjointWith rdf:resource="#dns_out"/>
    <owl:disjointWith rdf:resource="#dns_in"/>
    <owl:disjointWith rdf:resource="#nfsd_out"/>
    <owl:disjointWith rdf:resource="#smtp_out"/>
    <owl:disjointWith rdf:resource="#www_in"/>
</owl:Class>
<owl:Class rdf:ID="smtp_out">
    <rdfs:subClassOf rdf:resource="#Service"/>
    <owl:disjointWith rdf:resource="#dns_out"/>
    <owl:disjointWith rdf:resource="#www_out"/>
    <owl:disjointWith rdf:resource="#ftp_in"/>
    <owl:disjointWith rdf:resource="#smtp_in"/>
    <owl:disjointWith rdf:resource="#ssh_out"/>
    <owl:disjointWith rdf:resource="#ftp_out"/>
    <owl:disjointWith rdf:resource="#ssh_in"/>
    <owl:disjointWith rdf:resource="#www_in"/>
    <owl:disjointWith rdf:resource="#dns_in"/>
    <owl:disjointWith rdf:resource="#nsfd_out"/>
    <owl:disjointWith rdf:resource="#cfengine_in"/>
    <owl:disjointWith rdf:resource="#nfsd_out"/>
    <owl:disjointWith rdf:resource="#cfengine_out"/>
    <owl:disjointWith rdf:resource="#nfsd_in"/>
</owl:Class>
<owl:Class rdf:ID="ssh_in">
    <rdfs:subClassOf rdf:resource="#Service"/>
    <owl:disjointWith rdf:resource="#ssh_out"/>
    <owl:disjointWith rdf:resource="#cfengine_out"/>
    <owl:disjointWith rdf:resource="#dns_in"/>
    <owl:disjointWith rdf:resource="#cfengine_in"/>
    <owl:disjointWith rdf:resource="#smtp_out"/>
    <owl:disjointWith rdf:resource="#ftp_in"/>
    <owl:disjointWith rdf:resource="#dns_out"/>
    <owl:disjointWith rdf:resource="#nfsd_out"/>
    <owl:disjointWith rdf:resource="#smtp_in"/>
```

```
        <owl:disjointWith rdf:resource="#www_out"/>
        <owl:disjointWith rdf:resource="#nfsd_in"/>
        <owl:disjointWith rdf:resource="#www_in"/>
        <owl:disjointWith rdf:resource="#ftp_out"/>
        <owl:disjointWith rdf:resource="#nsfd_out"/>
    </owl:Class>
    <owl:Class rdf:ID="ssh_out">
        <rdfs:subClassOf rdf:resource="#Service"/>
        <owl:disjointWith rdf:resource="#nsfd_out"/>
        <owl:disjointWith rdf:resource="#ssh_in"/>
        <owl:disjointWith rdf:resource="#smtp_in"/>
        <owl:disjointWith rdf:resource="#smtp_out"/>
        <owl:disjointWith rdf:resource="#www_out"/>
        <owl:disjointWith rdf:resource="#ftp_in"/>
        <owl:disjointWith rdf:resource="#nfsd_out"/>
        <owl:disjointWith rdf:resource="#cfengine_out"/>
        <owl:disjointWith rdf:resource="#dns_out"/>
        <owl:disjointWith rdf:resource="#dns_in"/>
        <owl:disjointWith rdf:resource="#cfengine_in"/>
        <owl:disjointWith rdf:resource="#nfsd_in"/>
        <owl:disjointWith rdf:resource="#ftp_out"/>
        <owl:disjointWith rdf:resource="#www_in"/>
    </owl:Class>
    <owl:Class rdf:ID="Sunday">
        <rdfs:subClassOf rdf:resource="#Weekends"/>
        <owl:disjointWith rdf:resource="#Saturday"/>
    </owl:Class>
    <owl:Class rdf:ID="tcp">
        <owl:equivalentClass>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasFlag"/>
                <owl:allValuesFrom>
                    <owl:Class>
                        <owl:unionOf rdf:parseType="Collection">
                            <owl:Class rdf:about="#tcpack"/>
                            <owl:Class rdf:about="#tcpfin"/>
                            <owl:Class rdf:about="#tcpmisc"/>
                            <owl:Class rdf:about="#tcpsyn"/>
                        </owl:unionOf>
                    </owl:Class>
                </owl:allValuesFrom>
            </owl:Restriction>
        </owl:equivalentClass>
        <rdfs:subClassOf rdf:resource="#Protocol"/>
        <owl:disjointWith rdf:resource="#icmp"/>
```

```
    <owl:disjointWith rdf:resource="#udp"/>
</owl:Class>
<owl:Class rdf:ID="tcpack">
    <rdfs:subClassOf rdf:resource="#tcpFlags"/>
    <owl:disjointWith rdf:resource="#tcpsyn"/>
    <owl:disjointWith rdf:resource="#tcpfin"/>
    <owl:disjointWith rdf:resource="#tcpmisc"/>
</owl:Class>
<owl:Class rdf:ID="tcpfin">
    <rdfs:subClassOf rdf:resource="#tcpFlags"/>
    <owl:disjointWith rdf:resource="#tcpsyn"/>
    <owl:disjointWith rdf:resource="#tcpack"/>
    <owl:disjointWith rdf:resource="#tcpmisc"/>
</owl:Class>
<owl:Class rdf:ID="tcpFlags">
    <rdfs:subClassOf rdf:resource="#tcp"/>
</owl:Class>
<owl:Class rdf:ID="tcpmisc">
    <rdfs:subClassOf rdf:resource="#tcpFlags"/>
    <owl:disjointWith rdf:resource="#tcpsyn"/>
    <owl:disjointWith rdf:resource="#tcpfin"/>
    <owl:disjointWith rdf:resource="#tcpack"/>
</owl:Class>
<owl:Class rdf:ID="tcpsyn">
    <rdfs:subClassOf rdf:resource="#tcpFlags"/>
    <owl:disjointWith rdf:resource="#tcpfin"/>
    <owl:disjointWith rdf:resource="#tcpack"/>
    <owl:disjointWith rdf:resource="#tcpmisc"/>
</owl:Class>
<owl:Class rdf:ID="Test">
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#isPartOf"/>
            <owl:someValuesFrom rdf:resource="#Event"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <owl:disjointWith rdf:resource="#Event"/>
    <owl:disjointWith rdf:resource="#Level"/>
    <owl:disjointWith rdf:resource="#Variable"/>
    <owl:disjointWith rdf:resource="#DateTime"/>
    <owl:disjointWith rdf:resource="#Host"/>
</owl:Class>
<owl:Class rdf:ID="Thursday">
    <rdfs:subClassOf rdf:resource="#Weekdays"/>
```

```
        <owl:disjointWith rdf:resource="#Friday"/>
        <owl:disjointWith rdf:resource="#Wednesday"/>
        <owl:disjointWith rdf:resource="#Tuesday"/>
        <owl:disjointWith rdf:resource="#Monday"/>
    </owl:Class>
    <owl:Class rdf:ID="Tuesday">
        <rdfs:subClassOf rdf:resource="#Weekdays"/>
        <owl:disjointWith rdf:resource="#Friday"/>
        <owl:disjointWith rdf:resource="#Thursday"/>
        <owl:disjointWith rdf:resource="#Wednesday"/>
        <owl:disjointWith rdf:resource="#Monday"/>
    </owl:Class>
    <owl:Class rdf:ID="TwoDTS">
        <rdfs:subClassOf rdf:resource="#Test"/>
        <owl:disjointWith rdf:resource="#LDT"/>
    </owl:Class>
    <owl:ObjectProperty rdf:ID="TwoDTSType"/>
    <owl:Class rdf:ID="udp">
        <rdfs:subClassOf rdf:resource="#Protocol"/>
        <owl:disjointWith rdf:resource="#icmp"/>
        <owl:disjointWith rdf:resource="#tcp"/>
    </owl:Class>
    <owl:Class rdf:ID="Users">
        <owl:equivalentClass>
            <owl:Class>
                <owl:intersectionOf rdf:parseType="Collection">
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#hasInfluence"/>
                        <owl:someValuesFrom>
                            <owl:Class>
                                <owl:unionOf rdf:parseType="Collection">
                                    <owl:Class rdf:about="#cfengine_out"/>
                                    <owl:Class rdf:about="#dns_out"/>
                                    <owl:Class rdf:about="#ftp_out"/>
                                    <owl:Class rdf:about="#nfsd_out"/>
                                    <owl:Class rdf:about="#otherprocs"/>
                                    <owl:Class rdf:about="#smtp_out"/>
                                    <owl:Class rdf:about="#ssh_out"/>
                                    <owl:Class rdf:about="#www_out"/>
                                </owl:unionOf>
                            </owl:Class>
                        </owl:someValuesFrom>
                    </owl:Restriction>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#isOwnedBy"/>
```

```
                    <owl:allValuesFrom rdf:resource="#Client"/>
                </owl:Restriction>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#isOwnedBy"/>
                    <owl:allValuesFrom rdf:resource="#Server"/>
                </owl:Restriction>
            </owl:intersectionOf>
        </owl:Class>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="#Variable"/>
    <owl:disjointWith rdf:resource="#Process"/>
    <owl:disjointWith rdf:resource="#Diskfree"/>
    <owl:disjointWith rdf:resource="#Loadavg"/>
    <owl:disjointWith rdf:resource="#Service"/>
    <owl:disjointWith rdf:resource="#Protocol"/>
</owl:Class>
<owl:Class rdf:ID="Variable">
    <owl:equivalentClass>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#hasLevel"/>
                    <owl:allValuesFrom>
                        <owl:Class>
                            <owl:unionOf rdf:parseType="Collection">
                                <owl:Class rdf:about="#high"/>
                                <owl:Class rdf:about="#low"/>
                            </owl:unionOf>
                        </owl:Class>
                    </owl:allValuesFrom>
                </owl:Restriction>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#TwoDTSType"/>
                    <owl:allValuesFrom>
                        <owl:Class>
                            <owl:unionOf rdf:parseType="Collection">
                                <owl:Class rdf:about="#anomaly"/>
                                <owl:Class rdf:about="#dev1"/>
                                <owl:Class rdf:about="#dev2"/>
                                <owl:Class rdf:about="#microanomaly"/>
                            </owl:unionOf>
                        </owl:Class>
                    </owl:allValuesFrom>
                </owl:Restriction>
            </owl:intersectionOf>
```

```
        &lt;/owl:Class&gt;
    &lt;/owl:equivalentClass&gt;
    &lt;rdfs:subClassOf rdf:resource="&owl;Thing"/&gt;
    &lt;rdfs:subClassOf&gt;
        &lt;owl:Restriction&gt;
            &lt;owl:onProperty rdf:resource="#isPartOf"/&gt;
            &lt;owl:someValuesFrom rdf:resource="#Event"/&gt;
        &lt;/owl:Restriction&gt;
    &lt;/rdfs:subClassOf&gt;
    &lt;owl:disjointWith rdf:resource="#Level"/&gt;
    &lt;owl:disjointWith rdf:resource="#Test"/&gt;
    &lt;owl:disjointWith rdf:resource="#DateTime"/&gt;
    &lt;owl:disjointWith rdf:resource="#Event"/&gt;
&lt;/owl:Class&gt;
&lt;owl:Class rdf:ID="Wednesday"&gt;
    &lt;rdfs:subClassOf rdf:resource="#Weekdays"/&gt;
    &lt;owl:disjointWith rdf:resource="#Friday"/&gt;
    &lt;owl:disjointWith rdf:resource="#Thursday"/&gt;
    &lt;owl:disjointWith rdf:resource="#Tuesday"/&gt;
    &lt;owl:disjointWith rdf:resource="#Monday"/&gt;
&lt;/owl:Class&gt;
&lt;owl:Class rdf:ID="Week"&gt;
    &lt;rdfs:subClassOf rdf:resource="#DateTime"/&gt;
    &lt;owl:disjointWith rdf:resource="#Year"/&gt;
    &lt;owl:disjointWith rdf:resource="#Month"/&gt;
    &lt;owl:disjointWith rdf:resource="#DayOfWeek"/&gt;
    &lt;owl:disjointWith rdf:resource="#Minute"/&gt;
    &lt;owl:disjointWith rdf:resource="#Hour"/&gt;
    &lt;owl:disjointWith rdf:resource="#Second"/&gt;
&lt;/owl:Class&gt;
&lt;owl:Class rdf:ID="Weekdays"&gt;
    &lt;rdfs:subClassOf rdf:resource="#DayOfWeek"/&gt;
    &lt;owl:disjointWith rdf:resource="#Weekends"/&gt;
&lt;/owl:Class&gt;
&lt;owl:Class rdf:ID="Weekends"&gt;
    &lt;rdfs:subClassOf rdf:resource="#DayOfWeek"/&gt;
    &lt;owl:disjointWith rdf:resource="#Weekdays"/&gt;
&lt;/owl:Class&gt;
&lt;owl:Class rdf:ID="www_in"&gt;
    &lt;rdfs:subClassOf rdf:resource="#Service"/&gt;
    &lt;owl:disjointWith rdf:resource="#nfsd_out"/&gt;
    &lt;owl:disjointWith rdf:resource="#cfengine_in"/&gt;
    &lt;owl:disjointWith rdf:resource="#dns_in"/&gt;
    &lt;owl:disjointWith rdf:resource="#ftp_out"/&gt;
    &lt;owl:disjointWith rdf:resource="#dns_out"/&gt;
```

```
        <owl:disjointWith rdf:resource="#ssh_in"/>
        <owl:disjointWith rdf:resource="#ftp_in"/>
        <owl:disjointWith rdf:resource="#smtp_in"/>
        <owl:disjointWith rdf:resource="#cfengine_out"/>
        <owl:disjointWith rdf:resource="#ssh_out"/>
        <owl:disjointWith rdf:resource="#www_out"/>
        <owl:disjointWith rdf:resource="#smtp_out"/>
        <owl:disjointWith rdf:resource="#nsfd_out"/>
        <owl:disjointWith rdf:resource="#nfsd_in"/>
    </owl:Class>
    <owl:Class rdf:ID="www_out">
        <rdfs:subClassOf rdf:resource="#Service"/>
        <owl:disjointWith rdf:resource="#smtp_in"/>
        <owl:disjointWith rdf:resource="#ssh_in"/>
        <owl:disjointWith rdf:resource="#dns_in"/>
        <owl:disjointWith rdf:resource="#ssh_out"/>
        <owl:disjointWith rdf:resource="#cfengine_out"/>
        <owl:disjointWith rdf:resource="#cfengine_in"/>
        <owl:disjointWith rdf:resource="#nfsd_in"/>
        <owl:disjointWith rdf:resource="#ftp_out"/>
        <owl:disjointWith rdf:resource="#ftp_in"/>
        <owl:disjointWith rdf:resource="#smtp_out"/>
        <owl:disjointWith rdf:resource="#dns_out"/>
        <owl:disjointWith rdf:resource="#www_in"/>
        <owl:disjointWith rdf:resource="#nsfd_out"/>
        <owl:disjointWith rdf:resource="#nfsd_out"/>
    </owl:Class>
    <owl:Class rdf:ID="Year">
        <rdfs:subClassOf rdf:resource="#DateTime"/>
        <owl:disjointWith rdf:resource="#Week"/>
        <owl:disjointWith rdf:resource="#Month"/>
        <owl:disjointWith rdf:resource="#DayOfWeek"/>
        <owl:disjointWith rdf:resource="#Minute"/>
        <owl:disjointWith rdf:resource="#Hour"/>
        <owl:disjointWith rdf:resource="#Second"/>
    </owl:Class>
</rdf:RDF>
```

E.0.1

Bibliography

[1] *M. Burgess, H. Haugerud, T. Reitan, and S. Straumsnes. Measuring host normality.* ACM Transactions on Computing Systems*, 20:125–160, 2001.*

[2] *M. Burgess. Probabilistic anomaly detection in distributed computer networks.* Science of Computer Programming*, 60(1):1–26, 2006.*

[3] *M. Burgess. Two dimensional time-series for anomaly detection and regulation in adaptive systems.* Lecture Notes in Computer Science, IFIP/IEEE 13th International Workshop on Distributed Systems: Operations and Management (DSOM 2002)*, 2506:169, 2002.*

[4] *K. Begnum and M. Burgess. Principle components and importance ranking of distributed anomalies.* Machine Learning Journal*, 58:217–230, 2005.*

[5] *M. Burgess.* Analytical Network and System Administration - Managing Human-Computer Systems. *J. Wiley & Sons, Chichester, 2004.*

[6] *Mark Burgess. An approach to understanding policy based on autonomy and voluntary cooperation. In* IFIP/IEEE 16th international workshop on distributed systems operations and management (DSOM), in LNCS 3775*, pages 97–108, 2005.*

[7] *D. Aredo, M. Burgess, and S. Hagen. A promise theory view on the policies of object orientation and the service oriented architecture. In* submitted to Science of Computer Programming.

[8] *Thomas Biege. Virtual Burglar Alarm - Intrusion Detection Systems (Part 1)* $http://www.linuxsecurity.com/content/view/109769/169/$ *(accessed January 23rd, 2007)*

[9] *T.F. Lunt. A survey of intrusion detection techniques.* Computers and Security*, 12:405-418, 1993.*

[10] *A.K. Ghosh, A. Schwartzbard and M. Schatz. Learning Program Behavior Profiles for Intrusion Detection.* $http://www.usenix.org/event/detection99/full_papers/ghosh/ghosh_html/index.html$ *(accessed January 23rd, 2007)*

[11] *M. Burgess. Anomaly detection with cfenvd (accessed February 2, 2007)* $http://www.cfengine.org/docs/cfengine-Anomalies.pdf$

[12] *C. Kruegel, D. Mutz, W. Robertson, and F. Valeur. Bayesian Event Classification for Intrusion Detection* Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC 2003)*, 1063-9527/2003 IEEE.*

[13] *T. R. Gruber. A Translation Approach to Portable Ontology Specifications.* Knowledge Acquisition*, 5(2):199-220, 1993.*

[14] *M. R. Genesereth, and N. Nilsson.* Logical Foundations of Artificial Intelligence. *Morgan Kaufmann Publishers: San Mateo, CA. 1987*

[15] *R. Mizoguchi.* Tutorial on ontological engineering. *The Institute of Scientific and Industrial Research, Osaka University http://www.ei.sanken.osaka-u.ac.jp/pub/miz/Part1-pdf2.pdf [accessed March 22, 2007]*

[16] *Event Monitoring Enabling Responses to Anomalous Live Disturbances (EMERALD) http : //www.sdl.sri.com/projects/emerald/, [accesses April 16, 2007]*

[17] *J. Strassner. Knowledge Engineering Using Ontologies.* Handbook of Network and System Administration*, chapter Elsevier Handbook, 2007.*

[18] *P. Horn. Autonomic Computing:* IBM's Perspective on the State of Information Technology*, 2001.*

[19] *Gmez-Gaucha, H., Daz-Agudo, B., Gonzlez-Calero, P.A.:* Two-layered approach to knowledge representation using conceptual maps and description logics. *A. Caas F. Gonzalez Garca (Eds.): 1st International Conference on Concept Mapping, CMC2004, 2004.*

[20] *V. Raskin, C. F. Hempelmann, K. E. Triezenberg, and S. Nirenburg. Ontology in information security: A useful theoretical foundation and methodological tool. In* Proceedings of NSPW-2001*, pages 53 59. ACM, September 2001.*

[21] *J.L. Undercoffer, A. Joshi and J. Pinkston Modeling Computer Attacks: An Ontology for Intrusion Detection.* The Sixth International Symposium on Recent Advances in Intrusion Detection *In Proceedings LNCS-2516, 2003.*

[22] *R.A. Kemmerer and G. Vigna. Intrusion detection: A brief history and overview.* Security and Privacy, a Supplement to IEEE Computer Magazine*, 27-30, 2002.*

[23] *M. USCHOLD Building Ontologies: Towards A Unified Methodology* Proceedings of Expert Systems 96*,Cambridge, December 16-18th, 1996.*

[24] *Stanford Medical Informatics. http : //protege.stanford.edu/ [accesssed May 10th, 2007]*

[25] *N. F. Noy and D. L. McGuinness. Ontology development 101: A guide to creating your first ontology.* Technical Report SMI-2001-0880*, Stanford Medical Informatics, 2001.*

[26] *A. L. Rector. Modularisation of domain ontologies implemented in description logics and related formalisms including OWL.* In Second International Conference on Knowledge Capture (K-CAP)*, Sanibel Island, FL, 2003.*

[27] *M. Fernandez-Lopez and A. Gomez-Perez. Overview and analysis of methodologies for building ontologies.* The Knowledge Engineering Review, *Vol. 17:2, 129156, 2002.*

[28] *D. Calvanese, G. De Giacomo, M. Lenzerini, and D. Nardi. Reasoning in expressive description logics. In A. Robinson and A. Voronkov, editors,* Handbook of Automated Reasoning. *Elsevier Science Publishers (North-Holland), Amsterdam, 1999.*

[29] *P. Kazienko and M. Litwin On Using Topic Maps for Knowledge Representation. Information Systems Applications and Technology ISAT 2003 Seminar. newblock* Proceedings of the 24th International Scientific School. *Wroclaw University of Technology, 2003, pp. 100-107.*

[30] *P. Tetlow, J.Z. Pan, D. Oberle, E. Wallace, M. Uschold and E. Kendall. Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering.* $http://www.w3.org/2001/sw/BestPractices/SE/ODA/$ *accessed April 27, 2007.*