MINING FREQUENT SEMANTIC EVENT PATTERNS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ENİS SÖZTUTAR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2009

Approval of the thesis:

MINING FREQUENT SEMANTIC EVENT PATTERNS

submitted by **ENİS SÖZTUTAR** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Müslim Bozyiğit _____
Head of Department, **Computer Engineering**

Prof. Dr. İsmail Hakkı Toroslu _____
Supervisor, **Computer Engineering, METU**

Examining Committee Members:

Assoc. Prof. Dr. Ahmet Coşar
METU, Computer Engineering Department _____

Prof. Dr. İsmail Hakkı Toroslu
METU, Computer Engineering Department _____

Assist. Prof. Dr. Pınar Şenkul
METU, Computer Engineering Department _____

Assist. Prof. Dr. Tolga Can
METU, Computer Engineering Department _____

Güven Fidan
AGMLAB Inc. _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: ENİS SÖZTUTAR

Signature :

ABSTRACT

MINING FREQUENT SEMANTIC EVENT PATTERNS

Söztutar, Enis
M.S., Department of Computer Engineering
Supervisor : Prof. Dr. İsmail Hakkı Toroslu

September 2009, 99 pages

Especially with the wide use of dynamic page generation, and richer user interaction in Web, traditional web usage mining methods, which are based on the pageview concept are of limited usability. For overcoming the difficulty of capturing usage behaviour, we define the concept of semantic events. Conceptually, events are higher level actions of a user in a web site, that are technically independent of pageviews. Events are modelled as objects in the domain of the web site, with associated properties. A sample event from a video web site is the 'play video event' with properties 'video', 'length of video', 'name of video', etc. When the event objects belong to the domain model of the web site's ontology, they are referred as semantic events. In this work, we propose a new algorithm and associated framework for mining patterns of semantic events from usage logs. We present a method for tracking and logging domain-level events of a web site, adding semantic information to events, an ordering of events in respect to the genericity of the event, and an algorithm for computing sequences of frequent events.

Keywords: Web Usage Mining, Semantic Web Usage Mining, Semantic Events, Event Mining , Apriori

ÖZ

SIK ANLAMSAL OLAY DESENLERİ MADENCİLİĞİ

Söztutar, Enis
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. İsmail Hakkı Toroslu

Eylül 2009 , 99 sayfa

Özellikle Web'deki dinamik sayfa yaratımı ve zengin kullanıcı etkileşimi yüzünden, sayfa gösterimini temel alan geleneksel Web kullanım madenciliği metodlarının yararlılıkları sınırlıdır. Anlamsal olaylar kavramı, kullanıcıların davranışlarını modellemedeki zorlukları aşmak için geliştirildi. Kavramsal olarak, olaylar kullanıcıların daha yüksek seviyedeki eylemleridir ve sayfa gösteriminden teknik olarak bağımsızdır. Olaylar web sitesinin alanında ilgili özellikleri olan nesneler olarak modellenmiştir. Bir video izleme web sitesinden örnek bir olay, 'video izleme olayı', ilişkili özellikler ise 'video', 'videonun süresi', 'videonun uzunluğu', v.b. olarak verilebilir. Olay nesneleri, web sitesi ontolojisinin alan modeline ait olduğunda ise, olaylara anlamsal olaylar denilebilir. Bu çalışmada, web kullanım kayıtlarından anlamsal olay desenlerini çıkartacak yeni bir algoritma ve bir çerçeve önerilmiştir. Web sitesinin anlanındaki olaylarının takip edilmesi, kayıt altına alınması, anlamsal bilgi ile desteklenmesi, olayların genelliğine ilişkin bir sıralama ve sık olay dizilerinin hesaplanmasına ilişkin bir algoritma sunulmuştur.

Anahtar Kelimeler: Web Kullanım Madenciliği, Anlamsal Web Kullanım Madenciliği, Anlamsal Olaylar, Olay Medenciliği, Apriori

To my loving Mother and Father

ACKNOWLEDGMENTS

First of all, I would like to express my deepest gratitude to my supervisor, Prof. Dr. İsmail Hakkı Toroslu for his guidance, advice, criticism, patience and invaluable insight throughout the research.

I would also like to extend my thanks to my employer, Ekinoks Yazılım, for allowing me to pursuit this research, and giving me the opportunity to work on the web analytics product.

Last, but not least, I cannot truly exaggerate my gratitude and love for my parents for their support, encouragement and love.

This thesis, and the web analytics project is partly sponsored by the Türkiye Bilimsel ve Teknolojik Araştırma Kurumu (TÜBİTAK) / The Scientific and Technological Research Council of Turkey.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

1.1 Introduction

Previous work on analyzing web usage behavior is focussed on processing web server logs, due to the technicalities of the underlying architecture of the World Wide Web (HTTP protocol, web server logs, URL's, etc.). Typically, the data source for usage analysis is the pageview requests from the web server logs. Although the HTTP/1.0 protocol dictates a pageview based user interaction, usage of richer interaction models have exploded in the web using HTTP/1.1 connection keep alive, Adobe Flash, or AJAX. During the usage of the web site, a user hardly thinks of the web site as a set of pages, but rather she has higher level goals, such as finding a content, searching for something, listening to a song, etc. Pageviews are insufficient for capturing such information; therefore for mining rich and structured information, we need to keep the information to be mined at a more semantic level, similar to the work in [13, 39].

For overcoming the difficulty of capturing usage behavior, we define the concept of *semantic events*. Conceptually, events are higher level actions of a user in a web site, that are technically independent of pageviews. Events are modelled as objects in the domain of the web site, with associated properties. A sample event from a video web site is the 'play video event' with properties 'video', 'length of video', 'name of video', etc. When the event objects belong to the domain model of the web site's ontology, they are referred as *semantic events*.

In this work, we propose a new algorithm and associated framework for mining patterns of semantic events from the usage logs. We present a method for tracking and logging domain-level events of a web site, adding semantic information to events, an ordering of events in respect to the genericity of the event, and an algorithm for computing sequences of frequent

events.

We tested the proposed system on two web sites with very different domains. The first web site is a music streaming site, in which users search for, and listen to songs. The second is a web site of a large mobile network operator, with a rich content and high traffic. Resulting semantic event patterns from these studies have shown the approach to be useful and practical.

The work is part of an ongoing research and development project titled ”Web Analytics System” (Section 3.3). The project aims to build a scalable infrastructure for tracking, analysis, statistical and intelligent reporting of web usage behavior.

1.2 Organization

This text starts with a detailed introduction to the background concepts and related work. In Chapter 2, Web Mining, Description Logics, Semantic Web, and the so-called Semantic Web Mining fields are overviewed. In Section 2.4, current research frontier is referred and discussed in detail in respective sub-sections.

Chapter 3 covers the core concepts in this thesis. First some introduction to the work and motivation is given. Next, the project, which this thesis is a part of, is introduced and some project modules are explained. The chapter continues with the discussion of the concept of events and semantic events. Last, a two-phase algorithm for mining patterns from semantic event logs are given with sufficient formal detail.

Next, in Chapter 4, the experimental setup and the two web sites used for the experiments are introduced. The chapter continues with ontology definitions of the sites, the events and finishes with a discussion of the resulting patterns.

Last, the proposed approach is compared to some of the earlier efforts and a brief conclusion is given in Chapter 5.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this chapter, background knowledge relevant for this work is overviewed, and related work in the literature is summarized. In the first section, web mining is explained. The second section contains a short introduction to the Description Logics, which form the basis of the OWL-DL language. The next section continues with an overview of the Web Ontology Language. And finally in the last section, the intersection of semantic web and web mining fields is reviewed.

2.1 Web Mining

Web mining can be broadly described as the application of data mining techniques on the web data. The Web mining research is a converging research area from several research communities, such as Database, IR, and AI, Machine Learning and NLP [34]. The area is categorized in three subfields according to the type of the data to be mined. In web content mining, the data to mine is the the content of the web pages; in web structure mining, the data originates from the web link graph; whereas in web usage mining, the data to be mined is the usage logs of the web servers. In following sections, each subfield is summarized, with a special focus on web usage mining.

2.1.1 Web Content Mining

Web content mining describes the usage of knowledge discovery methods on web content. The content of the web can range from text, audio, video to semi-structured and structured data. HTML offers only a vague structure on the text, so the web pages in HTML are consid-

ered as semi-structured data. Examples of structured data, include extracted data from known HTML structures (such as forms, lists, etc), data in a self-describing formats (XML, JSON and data tables), and screen-scraped data from known web page structures. With the recent interest in machine-readable web, the human-readable content on the web is being slowly converted to a formal format, which will hopefully increase the portion of structured content on the web.

Much of the focus in the field was devoted to mining free-format text on the pages, which is abundant in the current web. The area enjoys a great body of work including information retrieval, search engines, topic discovery, classification and clustering of web documents, natural language processing, Multimedia Retrieval, finding patterns in text, extracting key phrases from documents, named entity recognition, multilevel databases, and web query systems. The methods can be categorized in two groups: the IR-centric views and the DB-centric views. In IR view, the documents are modelled as a *bag of words*, in *vector space model*. In DB-centric view, a structure is imposed over the raw data for enhancing query answering and information extraction. There also hybrid models combining web content and structure. The work in this field are not further discussed, as it is beyond the scope of this work. Interested readers can consult [18, 35, 34, 19].

2.1.2 Web Structure Mining

Hyper links in HTML documents form the basis of the data for web structure mining. The web is modeled as a directed graph, where the pages in the web are nodes and the hyperlinks from the pages are the edges [35]. Alternatively, instead of a node for every page, a node for every site can be generated, with a link from node A to node B, if there exists a page in the site A linking a page in site B. The most well known algorithms dealing with the web link graph include the Hyperlink Induced Topic Search (HITS) [33], Pagerank [40] and more recent adaptive Online Page Importance Calculation (OPIC) [1] algorithms, all of which are about finding the importance, or rank, of the web pages. In HITS, web pages are attributed with hub and authority scores. A good hub is a page pointing to good authority pages; and a good authority is a page pointed by good hubs. Thus the mutual recursive definition of a page's hub (h_u) and authority (a_v) score can be expressed as :

$$a_v = \sum_{u \to v} h_u$$

$$h_u = \sum_{u \to v} a_v$$

An iterative algorithm computes these scores from the web graph until they converge.

The pagerank algorithm assumes a random surfer model, in which a surfer starts with a page, and randomly clicks on the links on the page with probability $1 - p$, or randomly jumps to another page with probability p. Then a page's score(PR) is the probability of the surfer being in the page at a given time. The pagerank is formalized as :

$$PR(v) = \frac{p}{N} + (1 - p) \sum_{u \to v} \frac{PR(u)}{N_u}$$

, where N is the total number of pages and N_u is the number of outgoing links from page u, and $u \to v$ indicates page u links to page v.

Contrary to the above, OPIC is an online algorithm, meaning that it does not need to whole set of URL's to be fetched for computation. Every page is initialized with some amount of cash, and when a page is fetched, it's cash is evenly distributed to it's outlinks. The importance of the page ($I(v)$) is defined as the amount of cash, accumulated in the history of the page. More formally,

$$I(v) = \frac{H[v] + C[v]}{G + 1}$$

, where $C[v]$ is the current cash of page v, $H[v]$ is the accumulated credit history, and G is the sum of histories of all pages.

Other areas, where the structure of the web link graph is exploited include social network analysis, web search, supervised classification of page topics, discovering densely linked subgraphs, and spam detection. For a detailed survey, see [18, 35, 34, 19].

2.1.3 Web Usage Mining

Web usage mining (WUM) is the subfield of web mining with a particular focus on knowledge discovery from web usage data. Typically, the searched knowledge is either the frequent browsing behavior patterns, implicit relations among pages or user profiles. Frequent

browsing behavior is essential in understanding the navigation paths of the user, whereas relations among pages reveal latent relations between web pages, or the concepts that web pages present. User profiles are a key factor in categorizing the users, so that dynamic content can be generated depending on the profile. Learned knowledge of relations among pages, concepts or user profiles is utilized in web personalization, recommendation systems, site reorganization, business intelligence, and usage characterization [34, 47].

Various data mining algorithms are customized and applied to web usage data for knowledge discovery. In the following sub-sections, we overview some of the common usage mining methods. We then describe the data that is used in web usage analysis and the collection methods for usage data. Next, we discuss session management which is crucial in forming meaningful transactions. We end this section with a detailed explanation of the Apriori algorithm and variants.

2.1.3.1 Statistical Analysis

Statistical analysis is trivial, though it reveals precious insight. Statistical metrics are easily actionable, meaning that web site administrators can easily interpret the data at hand, and can make decisions relying on them. Example statistical reports include daily number of pageviews, daily number of sessions, daily number of unique visitors, daily number of visits to pages, top pages in the site, top entry/exit pages, distribution of browsers and their versions, etc. There are both commercial and open source projects that offer statistical analysis from web server logs.

2.1.3.2 Sequential Patterns and Association Rules

Discovering sequential patterns is one of the most important goals of web usage mining. Sequential patterns are inter-session sequences of URL's, that are frequently accessed. Each session contains ordered URL accesses. The sequential pattern analysis deals with finding frequent sequences of URL's that are accessed consecutively. An example 3-item sequential pattern is given below :

- 30 percent of users access page with URL `'/index.html'`,

- then the page with URL `'/academic/graduatePrograms.php'`,
- and then the page with URL `'/academic/graduateComputerScience.php'`.

There will be numerous patterns of the above form, so a *minimum support count* parameter is used to filter the patterns. The support of a pattern is defined as the number or sessions that contains the pattern. Only patterns which have greater support than the defined threshold is considered interesting.

Association rules offer another type of useful analysis. Association rules predict the user's behavior depending on which pages she visited. An example rule is depicted below :

- 80 percent of users who accessed page with URL `'/index.html'`,
- then access the page with URL `'/academic/graduateComputerScience.php'`.

Association rules can be computed from the sequential patterns in a straightforward computation. For each prefix subsequence of p of the sequence $s = \langle a_1, a_2, \ldots, a_n \rangle$, we can generate the rule $p \Rightarrow q$, where q is a suffix subsequence of s, and the last index of p is less than the first index of q.

A $confidence$ parameter is defined for association rules. Confidence for a sequential association rule $p \Rightarrow q$ is the support count for p, divided by the support count of pq, where pq is the concatenation of sequences p and q. More formally;

$$confidence(p \Rightarrow q) = \frac{support(p)}{support(pq)}$$

There are several ways in which discovered association rules and sequential patterns can be leveraged. Association rules can be used to predict a user's navigation path, from the current user session and links or advertisement can be recommended. For example, if a user's session matches with a head of a rule with high confidence, then the links in the body part of the rule can be used as predictors. Another way the sequential patterns can be helpful is web site reorganization (or web site adaptation), where navigation behavior can be analyzed to restructure site topology and page layouts.

In finding the sequential patterns, sequential pattern mining algorithms are employed. There are several algorithms in the literature, with varying characteristics. AprioriAll [3], GSP [46]

and SPADE [55], PrefixSpan, MEMISP, and SPIRIT are commonly used methods for this task. We will review AprioriAll in Section 2.1.3.8, and GSP in Section 2.1.3.9, since they are closely related to this work. For a more throughout survey, readers can consult [56].

2.1.3.3 Classification

Classification is the problem of assigning predefined labels to items. The labels are said to be classes of the items. The classification is a supervised learning method. In the web usage mining context, the users are classified into distinct profiles, so that actions specific to a profile can be applied. Profiles contain various parameters about the users, such as age, gender, or purchased items more than 1000 $. The profile information can be leveraged in recommendation and personalization systems.

2.1.3.4 Clustering

Clustering is the unsupervised method of finding related clusters and assigning items to these clusters. In the web usage mining domain, either users or pages are clustered [47]. Clustering of pages reveal the relations among pages and can be used in ontology learning, search engines, web assistance providers and link recommendation. Clustering of users can be performed to gain insight to similar navigation patterns, collaborative filtering, and link recommendation.

2.1.3.5 Web Usage Data

There are mainly two types of data used in the usage mining process, the HTTP access data from web servers and application specific usage data. In the former type, a web server keeps track of every HTTP request along with a selection of parameters passed with the HTTP header. Most of the work in the field deals with this type of data. The latter type, application specific data, is common in commercial application servers like WebLogic, BroadVision, and StoryServer. This type of data is logged in mostly customized formats for the task at hand, so no generic method for analysis exists.

The underlying protocols of the world wide web, especially the Hypertext Transfer Proto-

col (HTTP) enforces the user's access model to the resources, therefore the resulting data is closely tied to the protocol definitions. HTTP is basically a text based request-response protocol on top of TCP/IP. For every HTTP request, web servers keep track of various HTTP request and response headers. Typically, the ip of the client, the requested URL and the time for each request is essential in a usage analysis. Other possible fields for logging are listed in the web server logs part of the following section. HTTP does not offer an explicit mechanism for a connection-oriented communication. A stateful connection can be achieved indirectly by either keeping the state in the server side, or using cookies in the client-side to serialize the state information. If the state information logged by the server does not contain a session identifier, then sessions should be reconstructed using proactive strategies (Section 2.1.3.7).

In E-commerce applications, keeping track of business events and later analyzing the events for better insight has become indispensable. However, most of the time, the data collection method, type and format of the collected data is specific to the application. For example, an E-commerce application may track information about every transaction, including; (i) items purchased; (ii) user identifier; (iii) date and time; (iv) quantity; (v) total price; (vi) tax; (vii) shipping cost; (viii) billing country; (ix) billing city , etc, whereas a wiki site may keep track of every change in a document including; (i) the document title; (ii) user identifier; (iii) the size of the change; (iv) how many times the document is changed before; (v) how many changes the user has; (vi) is the user original author of the document , etc.

Another type of data which is gaining popularity is the logs of application level events. Application level events are distinct from HTTP access logs, in that, the events may not necessarily correspond to a URL request, and events are meaningful on their own. Events tracking is implemented in several major web analytics products, and they are being used for statistical event reporting, e-commerce analysis, and capturing goal conversions. Event tracking, and it's role in this project is detailed in Section 3.4.

2.1.3.6 Data Collection

Data collection is an important part of the web usage mining process. The technology for collecting the data ultimately defines the format, content, and the quality of the usage data. Not all types of data collection is suitable for a specific task. For example, in event tracking (see Section 3.4) we can only rely on Javascript tracking for a complete analysis. There are

mainly 4 ways of collecting web usage data [31, Ch. 2]: web server logs, web beacons, javascript tags, and packet sniffing. Below, these types are explained.

Using web Server logs is a convenient way to track user requests. The logs are generated by the HTTP servers for each user request. Web server logs is by far the most popular way to collect data in the academic context, however it has several limitations, especially related to re-constructing user sessions. Some common formats are used by today's web servers, which include the Common Log Format, the Combined Log Format and The Combined Log Format with Cookie [6]. In Combined Log Format with Cookie, the following fields are logged for each HTTP request; (i) client IP address; (ii) client authentication name; (iii) time of the request; (iv) HTTP request line, including HTTP method, the requested resource (URL), and protocol version; (v) HTTP status code; (vi) Size of the returned content; (vii) Referrer HTTP header; (viii) User-Agent HTTP header; (ix) Values of the client Cookies;

Web beacons are 1x1 pixel transparent images that are hosted by third-parties. The image is referred to (included) in the HTML source code of the page to be tracked. When the browser loads the image from the third-party this request is logged and later analyzed. This method is now obsolete in favor of javascript tagging which is superior [31, Ch. 2].

Javascript tagging is the insertion of some proprietary script (mostly in Javascript language) to the pages to be tracked. In hosted web analytics solutions, the script is hosted in the analytics vendor and it is referred in all the pages of the web site. When the user makes a pageview, the script is loaded and automatically executed by the user's browser. The script collects relevant data about the pageview and sends this data to the vendor's servers via another HTTP request.

A Packet sniffer is a hardware or software which analyzes the incoming packets, and logs the web server requests without interrupting the traffic. In some cases the sniffer can insert javascript tagging code to the response HTML for collecting more data. This method is more sophisticated to install than the previous methods and it is not very scalable in terms of increasing traffic, since all the packages should be routed to the packet sniffer.

Each method has it's share of advantages and disadvantages. Although using web server logs is traditionally the popular choice in academics, Javascript tagging is gaining momentum in the enterprise world, since it allows for service-oriented architectures, easier deployment, and better session management. For this work, we have utilized both methods, as detailed in Section 3.3.4 and 4.3.2.

2.1.3.7 Session Management

A session in the context of web browsing is the abstraction of a single user accessing a page, clicking on one of the links on the page, and so on until she moves to another URL by typing it, or closing the browser. The sessions are used in web usage mining as the main data form, since they correspond logically to the transactions in data mining.

However, constructing sessions from individual HTTP requests can be a major problem in web usage analysis. Since the HTTP protocol is stateless, the sessions should either be marked explicitly by the web server or sessions should be reconstructed using reactive methods. There are mainly two approaches to session tracking in practice: tracking user sessions with user cookies or reconstructing user sessions from raw server logs.

Reconstructing user sessions is a mature research area. Methods that use time oriented, navigation oriented, and and both [44]. Another method is to use both heuristics as well as the site's topology in the process to obtain more accurate (reconstructed) sessions [10]. In a recent study, the ontology of the web site is used in the process [32].

It is reported that session reconstruction with time and session based heuristics has an error rate of between 10% and 20% [44]. However, proactive methods using cookies effectively delegate session tracking to the browser, thus eliminating caching and session identification problems [47]. There are 3 cases where the sessions can be miscalculated in Javascript tagging. They are the cases when the user disables cookies, disables Javascript, or explicitly deletes the session tracking cookie during the session. It is reported in [23, Ch. 3, p. 36] that cookie accepting ratio is more than 99%, so depending on cookies is not considered as a major problem. Moreover cookie-based tracking is not vulnerable to the following problems :

- browser caching

- provider(server side) caching
- different computers using the same IP via NAT
- and using proxies

2.1.3.8 Sequential Apriori

Sequential Apriori is one of the mostly used algorithms in web usage mining. The algorithm is applied to web usage data to find frequent navigation patterns of the users. In the input data, each session is considered as a transaction and each URL in the session is considered an item (or itemset of size 1). The result of the algorithm is frequent patterns, which contains ordered list of URL's. Below, we outline the important parts of sequential Apriori algorithms.

In [3] the problem of mining sequential patterns over a set of customer transactions is given, and several related algorithms are proposed. The algorithms add the concept of sequential patterns to the previous Apriori algorithm [2].

In Sequential Apriori, a *transaction* has customer-id, transaction time and item purchased fields. An *itemset* is a set of items, and a *sequence* is an ordered list of itemsets. A sequence, s_1, *is contained in* another sequence s_2, if for every element of $a_i \in s_1$, there exists a $b_i \in s_2$, such that a_i is a subset of b_i, ($a_i \subseteq b_i$) and the index of b_i in s_2 is greater than the index of $b_{i-1} \in s_2$. An example from [3] is $\langle(3)(45)(8)\rangle \succ \langle(7)(38)(9)(456)(8)\rangle$, where $\succ$ denotes 'is contained in' relation. For every customer, a sequence is defined in the database, which consists of all the transactions of the user. The *support* for a sequence is defined as the number of customers whose transactions contain the sequence. In a set of sequences, a sequence s is maximal if s is not contained in any other sequence.

The problem is defined as follows; given a database of customer transactions, find the maximal sequences that have a greater support than a given threshold.

In [3], algorithms *AprioriAll*, *AprioriSome* and *DynamicSome* are introduced. AprioriAll counts all large sequences, whereas in AprioriSome and DynamicSome maximal sequences are counted first, then resulting sequences are counted in forward-backward phases.

AprioriAll algorithm resembles the Apriori algorithm, and iteratively runs 3 major steps:

1. Generate candidate sequences of length k (C_k) from the frequent k-1 length sequences (L_{k-1}), by a self join of L_{k-1}.

2. Scan all transactions to obtain candidate supports for C_k

3. Select sequences that have greater support than threshold (L_k)

A pseudo-code for AprioriAll is given in Algorithm 1. The algorithm starts with selecting large 1-itemsets, which is denoted L_1. Then each iteration generates the candidate set of length k, C_k, from the frequent itemset of length k-1, L_{k-1}. The candidate generation function joins the given set with itself, and returns the result. Two sequences, p and q of length k-1 are joined if their first k-2 elements are equal. The joined candidate is the concatenation of p with the last element of q. If the joined candidate sequence contains any subsequences of length k-1, which is not in L_{k-1} then, this candidate is deleted from the candidate set. At the next step of the algorithm, the dataset is parsed once, so that it is compared with the candidate set, and support counts for the candidates are incremented. Next, the candidates in the candidate set, C_k, whose support is greater than minimum support are selected for the set of frequent sequences, L_k. The algorithm returns the union of all frequent sequences.

Algorithm 1 AprioriAll

$L_1 \leftarrow$ {Frequent 1-sequences}
for k = 2; $L_{k-1} \neq \emptyset$; k++ **do**
  $C_k \leftarrow$ generateCandidates(L_{k-1})
  for all customer sequence, c in Database **do**
    Increment the count of all candidates in C_k that are contained in c
  $L_k \leftarrow \{c \in C_k | c.count \geq minSupport\}$
return Maximal sequences in $L = \bigcup_k L_k$

The AprioriSome and DynamicSome algorithms generate the candidates more greedily than AprioriAll, relying on the heuristic that counting non-maximal sequences can be skipped. However, AprioriSome and AprioriAll is shown to have similar performance characteristics [3].

2.1.3.9 GSP

In [46] Srikant and Agrawal introduced the Generalized Sequential Patterns (GSP) algorithm, which is a modified version of AprioriAll. The problem domain is enhanced to include time constraints, a rigid definition of transactions (sliding window) and domain taxonomies. Min-gap, Max-gap and window size time constraints respectively define minimum time between two data sequence items to be counted in the same itemset, maximum time between two data sequence items to be counted in the same itemset, and the maximum time of the sequence. Sliding window relaxes the definition of a transaction by allowing support count checking to be defined over a sliding window. *Is-a* relations are defined to form a DAG over the items, so that a the support for ancestor items can be incremented for their descendents in the database.

Unlike AprioriAll, the length of a sequence is defined as the number of items (not itemsets) in the sequence. The join step is a two-phase process with join and prune phases. A sequence s_1 joins with s_2 if the subsequence obtained by dropping the first item of s_1 is the same as the subsequence obtained by dropping the last item of s_2 The candidate sequence generated by joining s_1 with s_2 is the sequence s_1 extended with the last item in s_2. The added item becomes a separate element if it was a separate element in s_2 and part of the last element of s_1 otherwise [46]. In the prune phase, contiguous subsequences of length k-1 for a pattern are checked, and if any of them is infrequent, the candidate is deleted.

With the introduction of revised definitions of sequence and transactions, the support counting phase is different than AprioriAll. In GSP, supports are counted in a iterative forward-backward phases. The definitions of forward and backward phases are taken from [46]:

forward phase The algorithm finds successive elements of sequence s in data-sequence d, as long as the difference between the end-time of the element and the start-time of the previous element is less than max-gap, if the difference is more than max-gap, algorithm switches to the backward phase.

backward phase The algorithm backtracks and pulls up previous elements. If s_i is the current element and $endtime(s_i) = t$ the algorithm finds the first set of transactions containing s_{i-1} whose transaction times are after $t - maxgap$ The start time for s_{i-1} (after s_{i-1} is pulled up) could be after the end time for s_i. Pulling up s_{i-1} may necessitate pulling up s_{i-2} because the max-gap constraint between s_{i-1} and s_{i-2} may no longer be

satisfied. The algorithm moves backwards until either the max-gap constraint between the element just pulled up and the previous element is satisfied, or the first element has been pulled up. The algorithm then switches to the forward phase, finding elements of s in d starting from the element after the last element pulled up. If any element cannot be pulled up (that is, there is no subsequent set of transactions which contain the element) the data-sequence does not contain s.

The taxonomy over the items can be incorporated in GSP by enhancing each data-sequence, d, in the database with and extended sequence, d'. Each item in d is replaced by a it's set of ancestors. Then regular GSP is run over the database. Two optimization are possible in this context. First one is to pre-compute the ancestors, and drop ancestors which are not in any of the candidates. The second optimization is to not count sequential patterns that contain both the item and its ancestors.

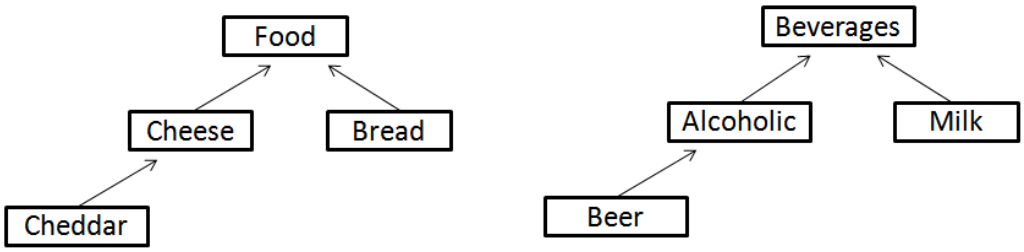

Figure 2.1: An example taxonomy for GSP

An example taxonomy is given in Figure 2.1. The taxonomy contains two top level classes, Food and Beverages. An example data set and its converted form are given in Table 2.1. The dataset contains two transactions. The first transaction contains two itemsets, each containing one element, and the second contains one itemset of two elements. The third column of the table contains the converted transactions, where for each item, it's ancestors in the taxonomy is added to the itemset it belongs.

Table 2.1: Sample Item-sets for GSP

Transaction		**Extended Transaction**
⟨⟨ Beer⟩, ⟨Cheese⟩⟩	→	⟨⟨ Beer, Alcoholic Beverages, Beverages⟩, ⟨Cheese, Food⟩⟩
⟨⟨ Bread, Cheddar⟩⟩	→	⟨⟨ Bread, Food, Cheddar, Cheese, Food⟩⟩

2.2 Description Logics

In this section we will briefly describe description logics, since they play a key role in giving semantics to the web. More specifically, the languages OWL-DL and OWL-Lite are based on the decades-long research in the field of Description Logics.

Description Logics (DL) refer to a family of Languages which are decidable fragments of First Order Logic. Description Logics have been developed as a formalism for knowledge representation(KR) [38]. The name *Description Logics* is originated by the fact that the domain of application is defined in concept *descriptions* and the languages differ from the earlier frame based and network systems by using *logic* as a basis for formal semantics [8].

In building the knowledge base, the concepts defining the domain are identified first. This part of the knowledge base is called the terminological part. Then the individuals and properties of individuals are defined. A Knowledge Base (KB) consists of two components: the TBox and the ABox [9]. The 'T' in TBox indicates that this is the *terminological* part, in which the vocabulary of the domain is defined. The letter 'A' in Abox stands for *assertional* part, in which assertions about named individuals in the domain are defined.

The vocabulary for the TBox is formalized with concepts and roles. Concepts are unary predicates and they are used to represent a set of (a class of) individuals. Roles, on the other hand, are binary predicates and represent relations among individuals. The language for building descriptions is a characteristic of each DL system, and different systems are distinguished by their description languages [38].

A knowledge base is depicted in Figure 2.2. The TBox for KB, consists of *terminological axioms*, whereas the ABox contains concept or role assertions about individuals.

Description logic languages specify model-theoretic semantics. A formal semantics for the DL is defined by the interpretation function $\mathcal{I}$. The domain of interpretation is denoted with

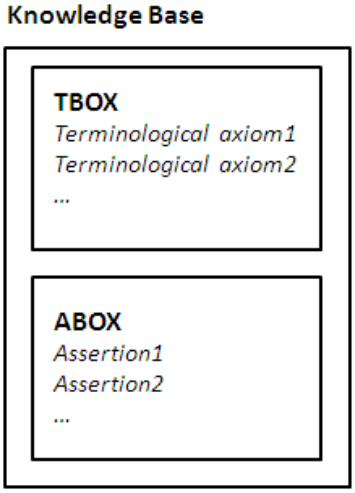

Figure 2.2: Knowledge Base

$\Delta^{\mathcal{I}}$ and consists of all the objects. The domain, $\Delta^{\mathcal{I}}$, together with the interpretation function, $\mathcal{I}$, defines the model for the DL. The interpretation function maps every concept to a set of objects in the domain and every role to a subset of cross product of domain with itself, i.e. for concept A, and role R :

$$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$$

$$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$$

DL systems offer algorithms for efficient reasoning support. Current DL systems support the so-called *tableau algorithms* for reasoning. The basic inference concept is *subsumption*, which defines whether a concept C, is more general than the other one D. The interpretation can be formalized as the set of objects for D in the domain is a subset of the set of objects for C, i.e. $D^{\mathcal{I}} \subseteq C^{\mathcal{I}}$. The other concepts include *satisfiability*, *equivalence* and *disjointness*. For every satisfiable concept C, there exists a model such that $C^{\mathcal{I}}$ is non-empty. Two concepts, C and D, are defined to be equal iff $C^{\mathcal{I}} = D^{\mathcal{I}}$ for every model of $\mathcal{I}$, and disjoint iff $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$ for every model of $\mathcal{I}$.

2.2.1 The $\mathcal{AL}$ Language

Description logics are named informally by the constructs they provide. The simplest DL is called $\mathcal{AL}$, which stands for attributive logic. Other languages are extensions to $\mathcal{AL}$. See [9] for the specific constructs of different languages, and [7] for a detailed listing for the naming conventions of DL's. Different languages embody different decidability properties, and there is a tradeoff between the expressivity and the complexity of inference on DL's.

Let A, B be atomic concepts and R a Role, C and D concept descriptions, $\top$ the Universal Concept and $\bot$ be the bottom concept . Then concept descriptions in the most basic language $\mathcal{AL}$ can be denoted as :

$$
\begin{array}{rll}
C, D \rightarrow & A| & \text{(Atomic Concept)} \\
& \top| & \text{(Universal Concept)} \\
& \bot| & \text{(Bottom Concept)} \\
& \neg A| & \text{(Atomic Negation)} \\
& C \sqcap D| & \text{(Intersection)} \\
& \forall R.C| & \text{(Value Restriction)} \\
& \exists R.\top & \text{(limited existential quantification)}
\end{array}
$$

Above descriptions of the language $\mathcal{AL}$ can be formalized using the interpretation function as:

$$
\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}} &= \emptyset \\
(\neg A)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \forall b.\, (a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\} \\
(\exists R.\top)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \exists b.\, (a,b) \in R^{\mathcal{I}}\}
\end{aligned}
$$

2.2.2 The $\mathcal{SHOIN}$(D) Language

The language $\mathcal{SHOIN}$(D) is of practical importance since the model theory of OWL-DL (Section 2.3.2) is based on this DL. $\mathcal{SHOIN}$(D) is a member of $\mathcal{SH}$ family of languages.

$\mathcal{SHOIN}$(D) adds the following constructs to the $\mathcal{AL}$ language :

- Complex concept negation ($\mathcal{C}$)
- Role hierarchy ($\mathcal{H}$)
- Transitive roles ($_{R^+}$)
- Union ($\mathcal{U}$)
- Unqualified number restrictions ($\mathcal{N}$)
- Nominals ($\mathcal{O}$)
- Datatype properties (D)
- Inverse roles ($^{-1}$)

For a complete syntax and semantics for the constructs of $\mathcal{SHOIN}$(D), please refer to [9, 53, 28].

2.3 Semantic Web

Today's web is mostly for human consumption. Although generated by automatic methods by machines, nearly all the content is prepared to be presented to users. Huge amounts of data and knowledge resides in text documents, or HTML documents, which cannot be processed easily by machines. A canonical example is the search engines. While very useful, search engines are based on keyword indexing and keyword matching to answer a user's query. Today's search engines do not truly attack the information retrieval problem, in which a user with a specific information need first transforms her request in a query syntax, sends the query to the information retrieval system, and obtains the information she needs. Instead, search engines accept a set of keywords with very restricted syntax, and returns a list of documents, that may contain relevant information.

Semantic Web, in basic, aims to augment today's web, with machine-processable data. In [14], Tim-Berners-Lee, et. al. define Semantic web as 'a new form of web content that is meaningful to computers'.

Semantic web vision is a layered approach. Each layer is meaningful on it's own, and is build upon the ones below it. The layers from the bottom to top are (i) Unicode/URI; (ii) XML + NS

+ XML Schema; (iii) RDF + RDF Schema; (iv) Ontology Vocabulary; (v) Logic; (vi) Proof; (vii) and Trust; .

In the base layer, Unicode allows a standard for exchanging symbols, and Uniform Resource Identifiers (URI) provide a naming schema for web resources.

Next, the Extensible Markup Language (XML) allows a syntax to mark data, while the schema for XML defines a grammar for the data. Namespaces (NS) are used to avoid name clashes in different documents. But namespaces can alternatively be used to refer to the same things in different documents.

The Resource Description Framework (RDF) offers a basic data model. It can be viewed as the first layer to introduce some semantics. RDF models contain resources, properties and statements. Resources can be anything with a URI. A statement in RDF is a triplet with a subject, property and an object. The subject is any resource, while the object is either a literal, a resource or another statement.

The RDF Schema (RDFS) defines a richer model on top of RDF. RDFS primitives include classes, properties, subclass and subproperty relations and domain and range restrictions. RDFS has it's own model-theoretic semantics and can be used as a simple ontology language.

The next layers, Ontology and Logic are considered together, since knowledge systems typically incorporate some way of logic reasoning. A language for defining ontologies, together with efficient inference support is the key to the Semantic Web.

Proof and Trust layers ensure, validity of obtained knowledge, which is essential in building fully automated systems. Not enough research has been done in these layers yet.

2.3.1 Semantic Web Components

2.3.1.1 Knowledge Representation

In giving semantics to web, several necessary components are identified. The first one is a canonical method for knowledge representation (KR). KR has been actively used and researched by the Artificial Intelligence community. In KR, knowledge, as perceived by humans, is converted to a form, that is parsable, storable, and processable by machines. KR in

Semantic Web differs from traditional systems in that, the Web (hence the Semantic Web) is distributed, heterogeneous, uncontrolled and the amount of information is huge.

2.3.1.2 Logic

Stored knowledge by machines is of little use, without a reasoning support. Adding logic to the knowledge allows inference about the knowledge at hand. Logic offers unambiguous formal semantics, in which everybody can agree on the meaning of a sentence. Moreover, reasoning systems exists for logic systems, which can infer new axioms, based on current knowledge base. To illustrate reasoning, suppose that all Pizzas have cheese in it, and AmericanHotPizza is a kind of Pizza. Then we can infer that AmericanHotPizza have cheese in it.

Logic languages can be quite expressive, however there is a tradeoff between the expressiveness of the language and efficient reasoning support. In general, complete and efficient reasoning support for logics can only be accomplished through limiting language constructs.

2.3.1.3 Ontologies

A definition of an Ontology is a specification of a conceptualization. In computer science, an ontology can be though as a Knowledge Representation system, offering a syntax for defining concepts in the domain, a common vocabulary, and support for reasoning.

In the context of the Semantic Web, ontologies are expected to play an important role in helping automated processes (called *intelligent agents*) to access information. In particular, ontologies are expected to be used to provide structured vocabularies that explicate the relationships between different terms, allowing intelligent agents (and humans) to interpret their meaning flexibly, yet unambiguously [28].

2.3.1.4 Agents

The real power of the Semantic Web will be realized when people create many programs that collect Web content from diverse sources, process the information and exchange the results with other programs [14]. Agents, takes goals, which are high level intentions, and

perform automated tasks, and return the results to the user. In determining the results, agents can search for information, talk to other agents, decide on what tasks to execute, and make reasoning. All of the available semantic technologies are expected to be leveraged by semantic agents.

2.3.2 OWL : Web Ontology Language

The Web Ontology Language (OWL) is a semantic markup language for publishing and sharing ontologies on the World Wide Web. OWL is developed as a vocabulary extension of RDF (the Resource Description Framework) and is derived from the DAML+OIL Web Ontology Language [52].

OWL is a family of languages for knowledge representation. The three sub-languages with increasing expressibility are : OWL-Lite, OWL-DL and OWL-Full. OWL-Full uses proprietary semantics compatible with RDFS, whereas OWL-DL and OWL-Lite is based on Description Logic (Section 2.2) reasoning. In fact OWL-DL and OWL-Lite can be viewed as expressive Description Logics, with an ontology being equivalent to a Description Logic knowledge base [28].

OWL-Full is the most expressive language. However, the language is so powerful that reasoning is undecidable [4]. In OWL-Full, the language constructs can be applied to each other.

OWL-DL is a restricted subset of OWL-Full. OWL-DL is based on $\mathcal{SHOIN}$(D) Description Logic Language. Application of language constructs to each other is disallowed in OWL-DL. Reasoning is complete, and decidable, at the expense of full RDF compatibility. In OWL-DL, a concept in DL is referred as a class, and a role is referred as an object property. A role with a data type range is referred as a datatype property. In this text, both the DL and OWL jargon is used interchangeably.

OWL-Lite is modeled on the $\mathcal{SHIF}$(D) language, and it is a sub-language of OWL-DL. Enumerated classes, disjointness statements and arbitrary cardinality is restricted in OWL-Lite. OWL-Lite is optimized for ease of use and implementation, however it is not used frequently.

Most of the elements of an OWL ontology concern classes, properties, instances of classes,

and relationships between these instances[51].

Classes A class is a set of elements (individuals). Classes group similar elements, with similar properties together. Every class in OWL, is a subclass of the top class, `owl:Thing`. Individuals belonging to a class are called instances of that class. The `rdfs:subClass` relation defines a hierarchy over classes.

Individuals Individuals are objects in the universe of things. Individual belongs to the classes. Individuals can be named, meaning that a unique URI is associated with the individual, or anonymous, having a null URI.

Properties Properties are binary relations. In OWL, two types of properties are defined: object properties and datatype properties. Object properties relate two individuals, while datatype properties relate instances of classes with RDF literals and XML Schema datatypes. These two are differentiated for the language to be decidable. The `rdfs:subPropertyOf` relation imposes a hierarchy over properties. The domain and range of a property can be specified with the `rdfs:domain`, and `rdfs:range` definitions, respectively.

Other Constructs Other language constructs in OWL include enumerations, boolean combinations of classes, transitive, symmetric, functional and inverse-functional properties, property restrictions, cardinality, and disjointness.

An OWL ontology, when it is serialized in RDF/XML is not very readable. An *abstract syntax* was developed to increase human readability, which is documented at [53]. OWL Abstract syntax uses an Extended BNF notation. Like the language, abstract syntax is rather complex. We give a subset of the abstract syntax for OWL-DL, as used in this text, in Listing 2.1.

An ontology in the abstract syntax is a set of directives, where a directive is either an axiom or a fact (ignoring Annotations from now on).

Facts can be either individual definitions or SameIndividual and DifferentIndividuals statements. In OWL-DL types can be general descriptions, however in this work, we only need classID's. Similarly, data literals can only be Unicode strings in normal form C. Readers should consult [53] for a complete reference of the abstract syntax and formal semantics for OWL.

Listing 2.1 A subset of OWL-DL abstract syntax

```
ontology  ::= 'Ontology(' [ ontologyID ]  directive  ')'
directive ::= axiom
            | fact
fact ::= individual
individual ::= 'Individual(' [ individualID ] { 'type(' type ')' }
               { value } ')'
value ::= 'value(' individualvaluedPropertyID individualID ')'
        | 'value(' individualvaluedPropertyID  individual ')'
        | 'value(' datavaluedPropertyID  dataLiteral ')'

type   ::= classID
```

2.4 Semantic Web Mining

Especially with the recent increase of interest in Semantic Web, integrating semantics to the web mining process is a natural direction for research. In [15], the authors elaborate on three ways that the fields of Semantic Web and Web Mining can co-operate. First, the web mining techniques can be used for creating or enhancing semantic information. Second, semantic information can be used in the web mining process, for better and more meaningful results. Lastly, data mining techniques can be applied directly to the vast amounts of structured semantic data to discover interesting relations and patterns. In the following subsections, we will review the three ways of integrating Semantic Web and Web Mining.

2.4.1 Extracting Semantics from the Web

The first way to integrate the two fields include methods for extracting latent semantics from existing knowledge on the Web. Web is in fact the de-facto and preferred platform for sharing information [41] and the information is abundant, . Thanks to the very nature of the Web, publishing data is easy and ubiquitous. However, the data in the web is unstructured or semi-structured at best, making it impossible for richer analysis. In this research direction, the existing data on the web is used to feed structured knowledge representation systems for learning knowledge organization, and populating ontologies.

Ontologies play a key role in giving semantics to the web. Ontologies are used as the main Knowledge Representation method on the web. However, creating, mapping, and managing ontologies is a cumbersome job. Although various tools exists, engineering an ontology is a specialty, in which the human in the cycle cannot be eliminated. However, semi-automated methods using Text and Web Mining for ontology development has been shown to be useful.

Ontology learning describes creating or improving an ontology through semi-automated methods. Semantic relations are extracted using combined techniques from Artificial Intelligence, Machine Learning, and Information Retrieval.

Ontology merging is another difficult problem the Semantic Web is faced with. In the semantic web vision, the (semantic) data sources are distributed in line with the de-centralized architecture of the web [14]. Ontologies are created by different groups with different interests, and slightly different understanding of the domain. A *survival-of-the-fittest* approach to public ontologies seems to be taking place in the current state of ontology development. Therefore, merging ontologies from different origins using traditional tools proves to be a hard task. On the other hand, several systems have been proposed for supporting the ontology engineer (see [50]).

Instance learning is concerned with the automatic or semi-automatic methods for populating the assertional part of the ontologies. Given an knowledge base, annotating existing data is necessary for the knowledge system to be useful. Semantic web can only be fully utilized, by annotating existing data as well as newly generated data. For example, without a method to give meaning, the huge amounts of text content in web pages can not be leveraged by logic systems.

2.4.2 Using Semantics for Web Mining

In the second form of integration, Semantic Web research can help Web Mining in giving structure to the underlying data to be mined. Adding semantic annotations for each type of data (content, structure and usage) allows richer analysis. However, once the web content is structured, the line separating content and structure mining diminishes. Here, we only detail the semantic web usage mining methods and recommendation. Interested readers can consult [50] for a detailed survey.

2.4.2.1 Semantic Web Usage Mining

Web usage mining can be enhanced by injecting semantics into various phases of the web usage mining process. The aim of usage mining is to better understand user's behavior. However, usage behavior is not fully captured in the logs of the web servers. For intelligently mining web usage, the usage information should be mapped to the semantic space.

In [13], the concept of *application events* is identified. Application events correspond to the higher level tasks defined in and specific to the domain of the site. They are categorized into three; content, service and complex application events. Content and service events are called atomic events, since they more or less correspond to pageviews. Complex application events are sequences of atomic events, and they are are usually described by regular expressions whose alphabet consists of atomic application events, or by an order structure on atomic application events.

Oberle et.al. [39] uses a semantic logging framework for retaining the full information of the request in logs. KA^2 can be described as a semantic portal, which converts URL requests to F-Logic syntax and routes to the inference engine, Ontobroker. The output is then written to HTML templates. They introduce a framework for semantic web logs. Each line in the log contains the URL, time stamp, and a feature vector. They define two types of feature vectors. In the first type, a dimension is defined for each concept and if the request (semantic) query contains that concept the value is set to 1. In the second type, a dimension is defined for every concept, attribute and relation. The log matrix can then be input to various web mining algorithms, such as clustering.

Recommendation and web personalization are other dimensions that could make use of semantic information. Building profiles from usage logs has been studied extensively in the literature. Usage profiles are generated through various web usage mining methods, and used in web personalization and recommendation. However, the profile information generally lacks structured information and reasoning support. Varying degrees of semantic knowledge, is incorporated for recommendation in a number of studies [21, 16, 42, 43, 20].

The concept of *domain level usage profiles* is introduced in [21] which is a weighted set of instances in the domain. More formally $pr = \{\langle o_1, w_{o_1}\rangle, \langle o_2, w_{o_2}\rangle, \ldots, \langle o_n, w_{o_n}\rangle\}$, where pr is the profile, o_i are objects in the ontology, and w_{o_i} are the weights of the objects in the profile.

Instances, or objects, belong to a class, C, and necessarily have the properties with the domain C. The property (or role) definition is extended to include an aggregation function, ψ_a, defined for each attribute, a. Using these definitions, *domain level aggregate profiles*, which represent aggregate data about usage, are computed. The profiles are first partitioned by classes, and these partitions are aggregated individually using the aggregate function, ψ_a. Two types of recommendations can be performed at this stage. First, the *current user profile*, which is the weighted items in the user browsing history are compared with the items in the ontology and the ones with similar features are recommended. Alternatively, current user profile can be matched with the aggregate domain-level user profiles and recommendations are performed from the aggregate profile.

In a recent study, [43], semantic information is exploited in processing the usage logs, and the recommendation system is fed with the resulting patterns. First the web server logs are converted to individuals in the site's ontology. Relevant pages in the web site are associated with one or more individuals. The ad-hoc mapping between the page URL's and individuals is stored for later processing. The SPADE [55] algorithm is enhanced so that taxonomical (sub-class hierarchy) information can be used in subsequence matching; i.e. an item in a sequence matches it's ancestors in the pattern. The algorithm outputs association rules, which are then used in the recommendation phase. Resulting association rules are from item sequences to item sequences. A window parameter is defined limiting the navigation history of the user for recommendation. When a user requests a page, the sequence of items (of length window size) in the user's history is checked with the association rules. The body part of the rules whose head matches are ranked in decreasing confidence. For the top item, a reverse lookup is performed on the page-to-item mapping to find the list of pages (URL's) that contain this item. At the last step, found pages are presented to the user as navigation recommendations.

2.4.3 Mining the Semantic Web

The third form of the intersection of Semantic Web and Web Mining, is the application of mining techniques directly to the structured data residing in the knowledge representation systems.

The closest field in the literature for mining structured information is the *Relational Data Mining* or *Inductive Logic Programming* [15]. In Relational Data Mining, patterns consisting

of more than one relation are searched in relational databases. Classification, regression, clustering, and association analysis are major techniques. However, recently attention has shifted from relational databases towards more meaningful, semantically-enabled systems, such as the ones in Semantic Web [30].

Józefowska et. al. propose a method for discovering frequent patterns from knowledge bases given in OWL-DLP (Descriptive Logic Programs) in [29], and improve the algorithm in [30]. Given a knowledge base in OWL-DLP, a set of patterns in the language of queries that contain a reference concept, and a minimum threshold, the task is to find queries with support greater than minimum support. The reference concept represents what is counted. A conjunctive query, $Q(x, y)$ is defined as a concatenation of $A(s)$ (atomic concept) or $R(s, t)$ (atomic role) atoms, where s and t are distinguished or non-distinguished variables. The support for a query is defined as an assignment of individuals in the ontology to distinguished variables. A level-wise algorithm, inspired by Apriori, is given in [29], which starts with the trivial query of the reference concept and gradually iterates by refining the query and counting supports. The algorithm is improved and a trie-based approach is given in [30].

CHAPTER 3

MINING SEMANTIC EVENT PATTERNS FROM WEB USAGE LOGS

3.1 Overview of the Chapter

This chapter starts with an introduction and motivation for the work. It continues with a detailed description of the web analytics project, which this work is a part of. Then the event tracking submodule is outlined, giving an introduction to the data collection with semantic aspects. At Section 3.5 the architecture for mining frequent semantic event sequences is discussed thoroughly with a formal statement of the problem and the algorithm.

3.2 Motivation

Web usage mining methods have been studied extensively and they have been proven to be useful to some extent. To date, especially in frequent sequential association rule mining, the bulk of the work uses URL's as the core data. However, this approach has several major limitations, which are listed below.

- URL's are hard to interpret on their own. Technical knowledge about the URL construction schema of the web site is generally necessary for meaningful analysis. Site analysts may not be closely connected to the web developers in some organizations.
- The query part of the URL include both relevant and irrelevant content, which is very hard to make use of in the usage analysis. Most of the work in this area discard the query part of the URL, however web pages in today's web are hardly static. Moreover,

if the query part is used, then some of the parameters may become problematic if they do not cause the web server to serve a different page. For example, consider the URL `'/search.jsp?query=computer&session=2342344&from=opensearch'`. The URL requests a page with the path `'/search.jsp'` and has 3 parameters in the query : `query`, `session`, and `from`. The first parameter is used as the query phrase, the second is for keeping the session id, and the last indicates that the user used OpenSearch for accessing the URL. Now, in the analysis, if the query part is erased, the user's searching for computers and something else cannot be differentiated, and information is lost. If we include the query part, then a different user with a different session id, but with same `query` and `from` parameters are erroneously counted as distinct. The problem becomes harder if we consider the `from` parameter. Does two URL's with different from parameters differ? To answer this question low level technical knowledge about the site domain and URL construction schema is needed.

- Not every action in the web page is captured by an HTTP request. With the invention of asynchronous web requests, extensive use of Javascript and Adobe Flash and dynamic layering of web pages, the concept of pageview is beginning to be blurred and hence becomes less relevant. For example in a video streaming site, a page request is logged, when a user requests for a page including the video. However, the actions of actually playing the video, stopping the video, skipping it, etc are not logged.

- URL's may not capture all the parameters about some user action. To illustrate, a user's preferences can be kept in a cookie, and the web server can decide on the content of the page using the URL and the cookie information obtained from the user.

- The assumption that every web page is about a more-or-less isolated content, and the aim of the web user is to satisfy her information need is hardly valid in todays web. Richer content pages, richer user interaction, and pages serving multiple purposes are abundant in Web. For example, we cannot associate any single content type, action or user goal for customizable web pages, which have several widgets (like Yahoo!'s home page or iGoogle page), or the social network sites (Facebook/Twitter content streams), RSS aggregators, etc. For these types of sites, a URL does not map to a unique page with deterministic content.

Due to the above limitations, traditional methods should be reconsidered for more correct and throughout analysis of web usage behavior. A natural direction is incorporating semantic knowledge in the process. In improving the usage analysis with semantic information, there are three requirements. First, a method should be devised to capture user intent more robustly than pageviews. Second, usage data should be captured or mapped to the semantic space. Last, an algorithm exploiting semantic relations should be devised according to the knowledge discovery task at hand.

As discussed in the introduction, we can effectively model the usage behavior with the concept of semantic events. Events in the application domain are discussed to be useful in enhancing mining results in [50]. In this work, events are tracked from the Javascript of the web site, and are thus independent of pageviews. It is possible to define one or more events for each pageview, and an event can occur at anytime during the viewing of the web page. Events are also more flexible than URL's, in that events are defined in the semantic space of the site's ontology. The literal and object properties of the events capture rich information about the event instance.

To intelligently mine the web usage data, the data to be mined or the supplementary data should be obtained in semantic form. There are mainly two methods for obtaining semantically enriched logs: converting existing logs to semantic form, or logging semantic information directly from the web server. Previous work, in the field of semantic web usage mining [13, 21, 16, 42, 43, 37] has mostly dealt with the former. In [39], the KA^2 portal, uses a semantic framework (RDF and F-Logic) to generate the web pages. As a part of the process, it logs the RDF annotations used to generate the site, forming a semantic log. However, both approaches depend on the assumption that every interaction with the site corresponds to a (single) pageview, and requesting a page signals an interest in several domain level objects. The assumptions does not necessarily hold for the reasons listed above. Our approach of logging semantic events asynchronously, and independent of the pageview is more relevant in today's web. Moreover, the metadata that is logged with each event contains all the interesting parameters, and nothing more.

In this work, we are interested in mining frequent patterns of users. For this task, we define the event and semantic events concepts, and provide an level-wise algorithm for discovering frequent sequences of semantic events.

3.3 Web Analytics Project

As stated in Chapter 1, Web Analytic project is an ongoing research and development project undertaken by Ekinoks Software Ltd. Şti., Ankara, Turkey. In this section a brief overview of the project is given. The work for the thesis heavily depends on some of the modules of the project, especially data collector and traffic analysis, therefore they are discussed in some detail in sections 3.3.4 and 3.3.5.

3.3.1 Project objectives

The objectives of the project can be summarized as

- Developing a tool for tracking users' requests for client web pages
- Developing algorithms for parsing and analyzing web request logs
- Producing reports for analyzing web usage behavior to be used by web site administrators
- Research for integrating semantic reasoning to web usage mining
- Research for rule based decision support systems to help the site admin
- Developing a system to analyze/compare user behavior for different web site designs (Web Site Test System)
- Developing a scalable storage, processing, querying and retrieval architecture for web usage logs and reports
- Developing a web based user interface for viewing reports and user interaction
- Developing a hosted web analytics service as a commercial product

3.3.2 Project Overview

Web analytics project offers traditional web analytics features, such as detailed reports about site's usage, Ad campaign tracking, A|B testing, etc. In addition to these features the project

aims to improve the analysis process by offering web usage mining, semantic web usage mining, and rule based decision support services.

A simplified view of the project architecture is given in Figure 3.1. There are 2 actors and 1 external component. The external component is the client web site (or shortly client) which is a client of the web analytics project. The first actor is the user who is using the client web site and she is mostly unaware of the web analytics service. The second actor is a manager or administrator of the client web site. She has the role of correctly deploying web analytics code to the site and consuming the reports generated by the web analytics project.

There project consists of 6 modules:

- Data Collector Module
- Traffic Analysis Module
- Web Site Design Test Module
- Semantic Web Usage Mining Module
- Decision Support Module
- Web Portal Module

Data collector module is responsible for tracking user's request to client web pages. The data collector module consists of a Javascript client and server side components for logging the requests. Client web sites include a reference to the Javascript source file, and make an explicit call via the exposed API for each page intended to be tracked. The data collector client then sends collected data to the data collector web servers. Data collector web servers save the data in local filesystem. A scheduled log fetcher service is run to dump the local data in the web servers to the distributed file system.

Traffic analysis module is responsible for parsing and analyzing the logs. The module consist of several algorithms, implemented as mapreduce jobs, to calculate statistical reports over the web usage data. The reports are then dumped to the database for later retrieval by the user facing components.

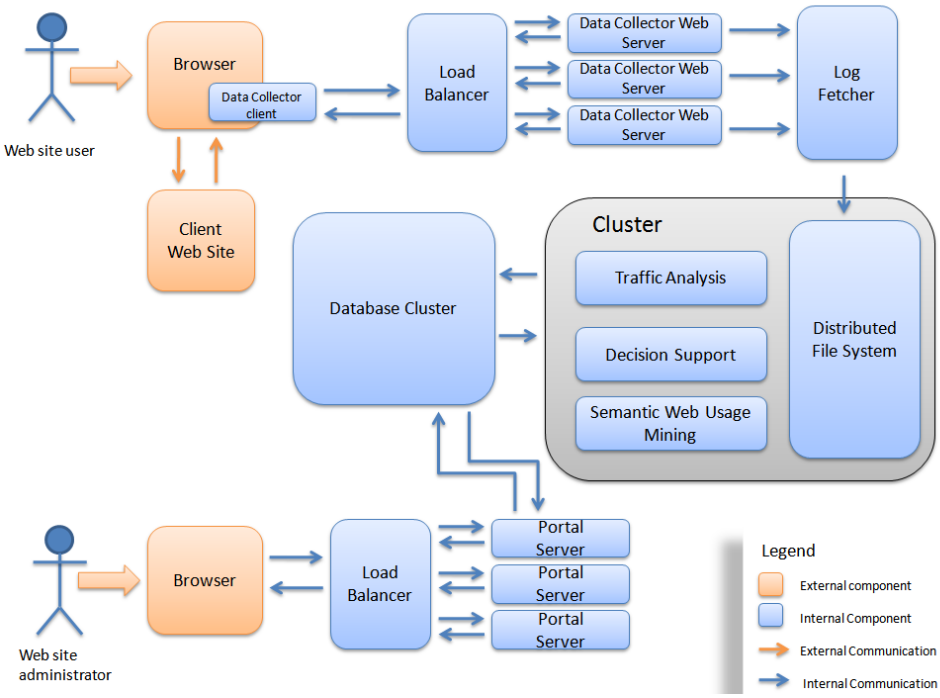

Figure 3.1: System Architecture Overview

Web site design test module includes the algorithms and user interfaces for comparing web site's statistical reports. This module allows A|B testing based on different times, different URL parameters or different javascript tags.

Decision support module is designed as a rule based system for supporting site administrator's decisions. A submodule of the decision support module calculates frequent browsing patterns for each client site for the given time interval. Frequent patterns are extracted from sessions using the sequential Apriori algorithm. A distributed version of the algorithm is implemented using Hadoop mapreduce. The patterns and optionally the web site topology are then used with predefined rules to suggest improvements in site navigation.

Web portal module is the main interaction point for interacting with client web site administrators and managers. The portal allows management of the service and exposes reports to the client.

In the following sections Apache Hadoop library, which is used as a distributed file system and mapreduce library is introduced and the Data Collector and Traffic Analysis module are

detailed. Semantic web usage mining module is build upon the work in this thesis, which is detailed in Section 3.5. As the other modules are rather orthogonal to the work, they are not further discussed.

3.3.3 Distributed File System and MapReduce

Apache Hadoop [5] is used as a distributed file system for storage and archival, and as a MapReduce [22] implementation. The distributed file system is used for reliable storage of all the data in the project. The traffic analysis, decision support, frequent pattern mining are all built on top of the mapreduce framework. Below very short descriptions are given, interested readers can consult [5, 54, 22, 24]

3.3.3.1 Distributed File System

Hadoop Distributed file system is modelled after the Google File System [24]. Hadoop distributed file system is scalable, reliable, and fault-tolerant with a single master, and hundreds or thousands of slaves architecture. All the metadata operations are handled by the master, which exposes a hierarchical namespace and file operations. Files are split into blocks, with a block size larger than traditional file systems (default 64MB). Slave nodes handle block operations and block storage and the data does not route through the master node.

3.3.3.2 MapReduce Implementation

Mapreduce is a simplified programming paradigm for processing large amounts of data in a large cluster. Mapreduce programs define custom map and reduce functions and the framework executes these functions over the data. Hadoop mapreduce allows writing highly scalable, fault tolerant, distributed and efficient programs with minimum coding.

3.3.4 Data Collector

In data collector sequence diagram (Figure 3.2), the sequence of communication between the client site's user, the client web site, load balancer and web analytics server is given. The

sequence of actions is as follows.

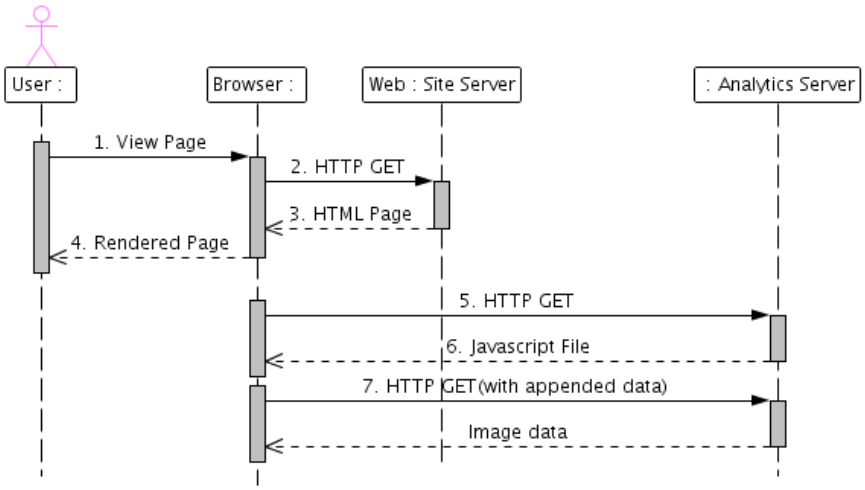

Figure 3.2: Data Collection Sequence Diagram

1. The user types in a URL or clicks a link in her browser
2. The browser makes an HTTP request to the client web site
3. The client web site serves the HTML page which contains a reference to the web analytics project javascript tracking client.
4. HTML page is rendered to the user (Asynchronous to the below steps)
5. The referred Javascript code is requested from the web analytics servers
6. Javascript file is returned from the HTTP request
7. When the script is loaded by the user's browser the script is executed. The script collects data about the request and makes an HTTP request to send the collected data to web analytics servers.

Collected data is transferred from the user's browser to the web servers by an HTTP request for an 1x1 transparent image file. The data is converted to parsable text and appended to URL of the image file as the query part. The following data is collected and logged per pageview request :

- The identifier for client web site
- Session identifier
- Visitor identifier
- URL of the requested page
- Title of the page
- Referrer
- Language as reported by the browser
- Color Depth
- Screen Resolution
- Whether Cookies are enabled
- Whether Java is enabled
- Adobe Flash Version is plugin is found
- Character Set as reported by the browser

The visitors are tracked using cookies. The script first queries for a specifically named cookie with the web analytics' domain. If the cookie is not found a new unique identifier is generated for the user and saved in the cookie. The cookie's expiration date is set to an unforeseeable future date, so that it practically never expires. If the cookie is found the visitor identifier is loaded and used. For the identifier 16 byte Universally Unique Identifier, Version 4 (UUID) [36] is used.

Web analytics project uses session tracking cookies and associated Javascript code for proactively tracking user sessions (see Section 2.1.3.6). For each pageview, a cookie is queried for previous session information. If the cookie is found, contained session identifier is used, if not a new session identifier UUID is generated and saved in the cookie. The cookie lifetime is set so that the browser deletes it once the session is finished.

3.3.5 Traffic Analysis

Traffic analysis module is the core module, which contains the business logic for web analytics. This module consists of piped jobs which are implemented in mapreduce, and thus runs distributed over several nodes. The first job parses the collected web server logs to extract the parameters send by the data collector. Both pageview and events are parsed and saved after the job in compressed, binary form. The second step constructs the sessions using the session identifiers. The records belonging to a session are grouped and sorted. The output of this job is a set of sessions. After the sessions are constructed, the statistical reporting jobs are run.

3.3.5.1 Data Cube Models

The web browsing data is modelled as 4 different data cubes with several dimensions and metrics. The dimensions are computed or filtered from the tracking data. The dimensions and the metrics for a sample data cube is given in Table 3.1. The data cube contains dimensions Browser Name, Browser Version, etc. and metrics Pageview Count, Visit Count, etc.

Table 3.1: Sample Data Cube

Browser Name
Browser Version
Operating System Name
Operating System Version
IP
Country
Screen Resolution
Referrer
Referrer Search Engine
...
Pageview Count
Visit Count
Total Time on Site
...

3.3.5.2 Materialized Cuboids for Reports

From the data cube we can define a lattice of cuboids. Cuboids are defined as data cubes with different levels of summarization [27]. For example a cuboid with dimensions Browser Name and Browser Version, and all the other dimensions rolled up will result in Browser Name x Browser Version x Pageview Count table.

Similar to the above example, some of the data cuboids in the lattice are selected as a basis for the reports. The data cubes, dimensions and the metrics are then computed and aggregated accordingly to generate the final reports. The data cube model and and the lattice is a conceptual data model employed in traffic analysis module. However it is not space and time efficient to generate all the cuboids in the lattice. In reality the base cuboid with all the dimensions is summarized for a few selected dimensions, and only cuboids with one, two or three dimensions are materialized and dumped into the database. The business requirements for the web analytics project determines which dimensions are materialized. Below a sample list of reports are given :

- Daily pageview count, visit count, average time on site, and new visits for operating systems between given time intervals
- Daily pageview count, visit count, average time on site, and new visits for countries between given time intervals
- Daily pageview count, visit count, average time on site, and new visits for referrer pages between given time intervals
- Daily number of visit counts for visitors which visit the site 1, 2, 3, 4, 5..9, 10..19, 20, 49, 50+ times
- Daily percentage of link clicks for each link on the home page

3.4 Event Tracking

Event tracking is a part of the Data Collector module (Section 3.3.4), however due to its major role in the semantic web usage mining work, it is discussed here, separately.

3.4.1 Events

Events are conceptual actions that the user performs to achieve a certain affect. Events are used to capture business actions that are defined in the site's domain. Some examples of events are 'Play' event for a video on the page, 'Add to shopping cart' event for and E-Commerce site, or 'add friend' action for a social site. Most of the time, events have associated properties that define the event. For example a 'search event' has the search query as a property, and 'play a song' event has a property showing which song is played.

With the advent of the rich Internet applications leveraging Adobe Flash, Microsoft Silverlight or asynchronous XML/JSON communication, tracking and reporting domain events along with pageviews has emerged as a special need. Event tracking is implemented in a few web analytics projects [31, Ch. 12],[25], however event tracking is considered only an extension to pageview based reporting. As an example let us look at how event tracking is implemented in Google Analytics.

Event tracking in Google Analytics[26] is done via explicit Javascript calls. There are 2 required and 2 optional attributes for defining the event parameters. The required event attributes are category, which is the name of the group of objects, and the action name. The label attribute is optional and defines an additional dimension. The value is again optional and used for specifying numerical data in integer type. The names for actions are treated as unique, and reports for the actions with the same name are grouped together. The category and label attributes are used as dimensions for the reports. The number of events, the average and sum of the value attributes are reported as metrics.

3.4.2 Event Tracking in Web Analytics Project

Similar to it's competitors, events play a key role in the web analytics project. However, unlike the others, web analytics project enables a more flexible and customizable architecture for event definition and event parameters. In the Web Analytics Project model, every event is defined as an object. Objects can have two types of properties. A property of an object relates the object to either another object, or a data literal. In this simple event model, capturing arbitrary detail about an object becomes possible. The total information about an event is the event object graph, including the event object itself, and it's related objects recursively.

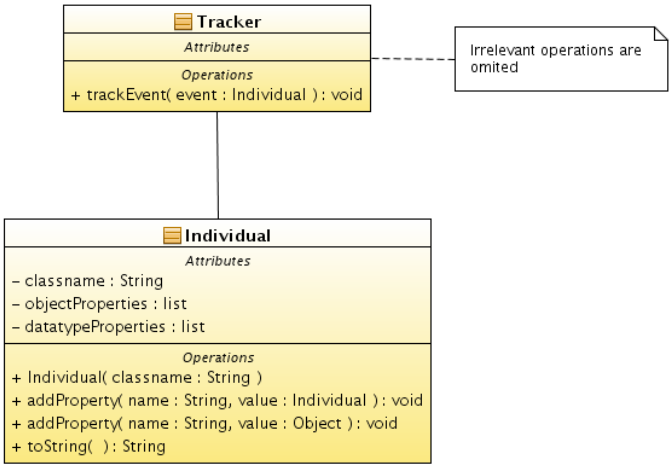

Figure 3.3: Event Tracking Class Diagram

Events in web analytics project are tracked and logged from the Javascript client, which enables logging events from Ajax and Adobe Flash applications. The Javascript client of the Web Analytics project exposes a simple Event Tracking API for its users. The simplified URL class diagram is given in Figure 3.3. The API defines a class, which is named `Individual`. The `Individual` class has a `classname` attribute and `addProperty(name, value)` methods. An application event is captured by an instance of an `Individual` object, which can have multiple properties. A property for an `Individual` is a name, value pair. The name is a string name and the value can either be one of the Javascript primitive data types(numbers, strings, boolean values), or another `Individual` object. The `Tracker` defines a `trackEvent()` method which takes an object of type `Individual`. This method is responsible for serializing and logging the `Individual` object, and its dependent objects. A check is performed to avoid cyclic dependencies in the object dependency graph.

Events in the domain of the web site is composed using the API defined above. Every event is represented with an Individual object, however all the Individual objects need not correspond to an event. The classname of the Individual object defines the corresponding classname in the ontology of the web site. The objects of type `Individual` can have named properties which are either literals, or other objects. The data associated with each event is an `Individual`

object, and its properties, possibly including other objects. Therefore an event is actually a set of individuals having literal properties and related to each other with object properties in a tree topology. The structure of the event, and the mapping of events into semantic space is discussed more rigorously in the following section.

An example invocation to track an event is given in Listing 3.1. The example is taken from the source code of a music streaming site [17], which is detailed in Section 4.2. The Javascript code is run whenever a song is played in the site, thus it captures a 'play song event'. The code uses the provided API to construct the event objects. In line (1) and (2), two objects of type `Individual` are created having `classname` attributes 'PlaySongEvent' and 'Song', respectively. Line (3) adds a name property to the song individual. Line (4) adds a song property to PlayVideoEvent individual, relating it with the Song individual. At this stage, the event tree (detailed in the next section) is complete. Finally the event is tracked in line (5).

Listing 3.1 Javascript code for tracking a sample event

```
(1)  var evt = new tracker.Individual('PlaySongEvent');
(2)  var song = new tracker.Individual('Song');
(3)  song.addProperty('name', name);
(4)  evt.addProperty('song', song);
(5)  tracker.trackEvent(evt);
```

All the data about an event is logged by the `trackEvent()` function of the `Tracker` class. Actual sending of the data is performed by collecting required data, converting it to string and sending the data via an HTTP request. The data collected and logged along with the individuals per event is listed below.

- The identifier for client web site
- Session identifier (see Section 2.1.3.7)
- Visitor identifier
- Client IP address
- URL of the page
- Time of the event

- Character Set as reported by the browser
- Individuals and properties in the event

Event tracking is tightly coupled to the semantic web usage mining framework introduced in this thesis. Events and pageviews are considered to capture all the aspects of user's interaction with the site. This assumption is dependent on the correct and sufficient usage of the event tracking API by the site's administrator. The interactions and pageview requests must be explicitly marked by the administrator, and the API functions should be called for every page and event to be tracked. Although this may seem a disadvantage of script based log collection, wide adoption of the technique by major web analytics and large number of deployments proves that it practical.

3.4.3 Events as semantic objects

As discussed above, and detailed in Figure 3.3, events are captured by instances of Individual objects. This allows capturing events in arbitrary detail. Moreover the event objects created by the client web page effectively model the domain of the users action. With a correspondence between the ontology capturing the user's action and the event objects, we can treat the events as semantic entities, and call them *Semantic Events*.

The responsibility of construction of the ontology and event objects, and tracking the events (by calling the API) in the appropriate context is delegated the site's administrator. The motivation for this is to offer the framework to all of the parties, and to allow usage of this feature by interested sites. The site's ontology and related events are best understood by the people who create the content for the site. For this work, we have developed the ontologies and implemented event tracking for two web sites. Please refer to Chapter 4 for more information.

Events in the web site are expected to have a type. That is, there are expected to have a number of distinct types of events with varying parameters. For example, for a video viewing site a subset of the possible events are Play Video, Stop Video, Search, Login, etc. and for each event, corresponding classes, properties and linked classes are defined. When some event occurs, the individuals are created and logged with appropriate values. For example the Play Video event might have a PlayVideo individual, a video property linking a Video individual, which in turn has several properties such as length, id, name, etc. However, the assumption of

types of events is not a necessary condition for the frequent pattern mining algorithm. Events can be defined partially, by only listing known properties. For example if the length of the video is not knows for some of the videos, it can be omitted.

Since the event object dependency graph is non-cyclic, there is a root, and objects can only be linked to one object, the graph of event objects can be considered as a tree structure. Three types of nodes are defined: nodes containing an individual, a datatype property or an object property. The properties of an individual are stored as children. A datatype property node is always a leaf node and contains the property name and value. An Object property node contains the property name and has a single child, which is another individual node.

An illustration for the tree structure is given in Figure 3.4. The knowledge base for this event include classes `Video`, `PlayVideoEvent`, `KittenVideos`, `VideoCategory`, and `User`, datatype properties `Title`, `Tag`, `ID` and `Length` which all have `Video` as domain, `username` and `isSubscriber` with `User` as domain, and object properties `video` with domain `PlayVideoEvent` and range `Video`, `Category` with domain `Video` and range `VideoCategory`, and `Submitter` with domain `Video` and range `User`. The `KittenVideos` class is a subclass of `VideoCategory`. When a user views a video with the title ”mykitten”, and event is created with an individual of class `PlayVideoEvent`, as the root of the tree. The individual has a single object property child, video, which relates the individual to an individual of type `Video`. The `Video` individual has several datatype properties such as the `title`, `ID` and `Length` of the video. It has also several `Tag`s. The category object property relates this video to an individual of type `KittenVideos`. And the `Submitter` property relates a `User` individual with `username` and `isSubscriber` properties. Please note that, even when the user views the same video, different individuals for `Video`, `User`, and `KittenVideos` are created, since there is no way to refer to the definitions in the ABox from the event tracking client. However, similar events will be aggregated appropriately using anonymous objects.

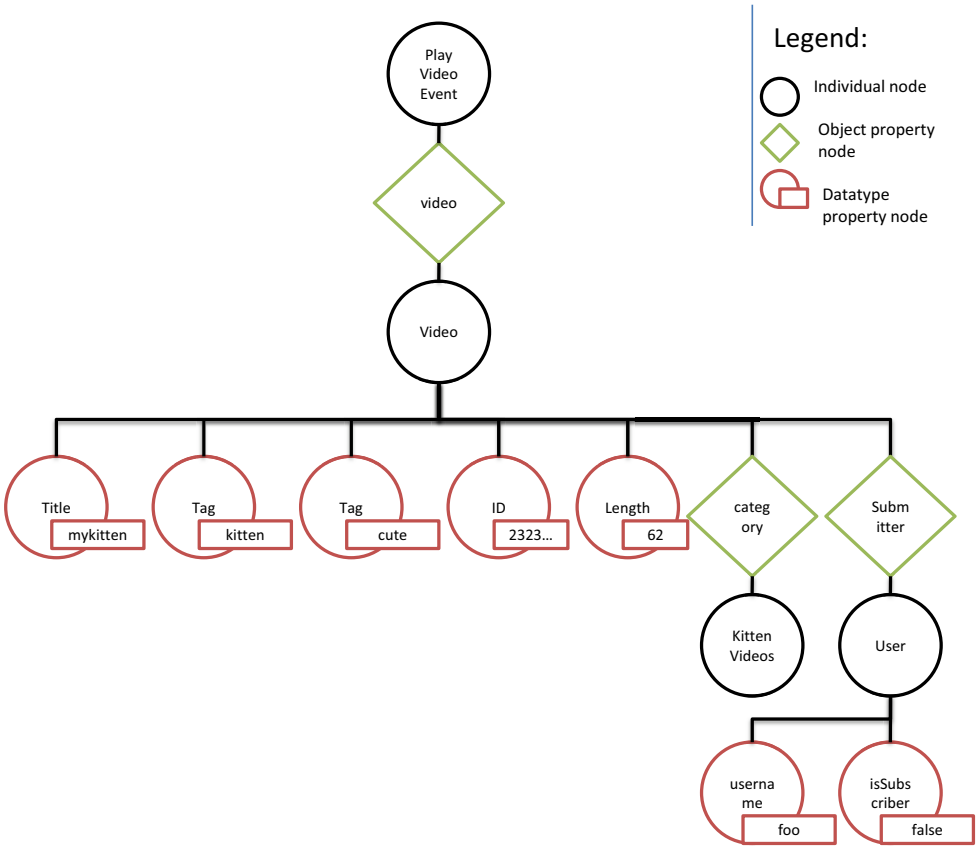

Figure 3.4: Event Diagram

To treat events as semantic entities we define formal semantics in Section 3.5.1.2, where atom and atom-trees are defined. Events are formalized with atom-trees, and individuals and properties are formalized with atoms. By a mapping between tracked events and their semantic counterparts, we are able to perform mining tasks leveraging semantic inference.

The mapping between the event objects and the semantic objects are done using the `classname` attribute, and the names of properties of `Individual` objects. These names are matched with the class and property names in the given web site ontology, and an exact match is searched. The pseudo code for the matching algorithm is given in Algorithm 2. The algorithm assumes that there is a given ontology capturing the event and property definitions. The algorithm takes the ontology model and the XML namespace for the ontology. The function

`getMapping()` takes an event, which is generated by the site and returns an `Individual Atom`. The `IndividualAtom`, `DatatypePropertyAtom` and `ObjectPropertyAtom` classes extend the `Atom` class, and hold ontology class, datatype property and object property definitions respectively.

Algorithm 2 Event Ontology Mapping

function *getMapping*(*e* : Event) : IndividualAtom

```
Ont ← Ontology given in OWL-DL
NS ← XML namespace for the ontology
if ∃ an OntClass ∈ Ont with NS + e.classname as URI then
    ontClass ← 0nt.getOntologyClass( NS + e.classname )
    ind ← IndividualAtom(ontClass)
    for all property p of e do
        if e is a datatype property then
            if ∃ a datatype property ∈ Ont with URI NS+p.name as URI then
                add DatatypePropertyAtom( p.name, p.value ) to ind as a child
            else
                return NIL
        else if e is an object property then
            if ∃ a object property ∈ Ont with URI NS+p.name as URI then
                c ← getMapping(p.value)
                if c ≠ NIL then
                    add ObjectPropertyAtom(p.name c) to ind as a child
                else
                    return NIL
            else
                return NIL
    return inv
else
    return NIL
```

3.5 Frequent Pattern Discovery from Semantic Event Logs

The events are tracked via the Event Tracking API and they are saved by the Data Collector Module. These logs are called semantic event logs, since they contain semantic events. Then the traffic analysis module, parses the logs, and groups these events in sessions. The sessions files becomes the input to the frequent pattern discovery phase.

In the next phase, frequent pattern discovery from semantic event logs, the background knowledge is given in a Description Logic Knowledge Base (KB). We use OWL-DL as a knowledge representation Language. The ontology (or KB) is taken as an input, which contains only terminological axioms. The class and property axioms are read from the ontology definition file (OWL File) and a memory model is built using the Jena library. Any definition regarding individuals in the ontology is discarded. The assertional part of the ontology is given in the event logs. Although the events in the event log are mapped to the semantic objects in the ontology model, no individual axiom is added to the Jena memory model of the ontology.

This section starts with formal definitions and problem statement. In the next section, the proposed algorithm is explained in detail and an optimization for filtering subjectively uninteresting patterns is given.

3.5.1 Problem Statement and Formal Definitions

In this section formal definitions for the web usage mining task is given. The formal definitions for the traditional setting is given first, then they are adopted and extended in the next section to form a mathematical language for the algorithms.

3.5.1.1 Frequent Pattern Discovery from Web Server Logs

In the traditional setting for web usage mining, the problem definition is a follows.

Let U denote a i^{th} user, P a URL representing a pageview, and S a session of the user. A session is a set of ordered pageviews by a certain user :

$$S =< P_1, P_2, P_3, \ldots >$$

, where $< \ldots >$ denotes an ordered set, and $S[i]$ denotes the i^{th} element of the ordered set. The data set, D for web usage mining is a set of sessions :

$$D = \{S_1, S_2, S_3, \ldots\}$$

, where $\{\ldots\}$ denotes an unordered set

Then the aim of frequent pattern mining algorithm is to find the set of frequent patterns, L, where a pattern Q is defined as :

$$Q =< P_1, P_2, P_3, \ldots >$$

$$L = \{Q_1, Q_2, Q_3, \ldots\}$$

A session S contains, or supports, the pattern Q iff Q is a subsequence of S. Q is a subsequence of S if Q can be obtained by deleting some elements of S. More formally:

$$S =< P_1, P_2, P_3, \ldots, P_n >$$
$$Q =< P'_1, P'_2, P'_3, \ldots, P'_m >$$
$$\text{Q is a subsequence of S} \iff \forall i \in [1, m], \exists j_i \text{ s.t. } Q[i] = S[j_i] \wedge j_i > j_{i-1}$$

The support of the pattern is the the number of sessions containing that pattern. Frequent patterns are the patterns which has greater support than the minimum support, which is a parameter to the algorithm.

3.5.1.2 Frequent Pattern Discovery from Semantic Event Logs

In mining frequent semantic events, we make similar definitions as the above section. Events were defined in section 3.4.3, however a more formal definition is given here. Events, hereby referred as atom trees, contain nodes which are called atoms. The aim of the algorithm is to find frequent atom trees.

Definition 3.5.1 (Atom) *An atom is either an individual of class C, a datatype property assertion, or an object property assertion. An atom, denoted with α, is a node in the atom-tree (defined below).* ■

More specifically, for the knowledge base, KB, interpretation function $\mathcal{I}$, domain of interpretation, $\Delta^{\mathcal{I}}$, domain of data types, ${\Delta_D}^{\mathcal{I}}$, Concept C, Role R, individual o_i, o_j data values, v_j,

and U for Datatype Role ($U^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times {\Delta_D}^{\mathcal{I}}$)

$$\begin{aligned} \alpha &= C(o_i) \\ &\mid U(o_i, v_i) \\ &\mid R(o_i, o_j) \end{aligned}$$

The property assertions have the parent individuals as the object. The object property atom have the child atom's individual as the subject of the statement. Individuals in the Knowledge Base can be unique or anonymous. The unique individuals have a unique URI, whereas the anonymous individuals does not. The anonymous individuals are used as representative samples of their class.

Definition 3.5.2 (Atom-tree) *An atom-tree is a tree of connected atoms. The atom tree represents a domain event in the web site's ontology.* ■

An atom tree, denoted with a, is formalized as :

$$a = \{\alpha_1, \alpha_2, \alpha_3, \ldots\}$$

The atoms in the atom tree are connected in a tree structure. If $\alpha_i = R(o_i, o_j)$, than $C(o_i) \in a, C(o_j) \in a$, and if $\alpha_i = U(o_i, v_i)$, than $C(o_i) \in a$

Definition 3.5.3 (Session) *A session is an ordered set of domain events of a single user in a certain browsing activity.* ■

Unlike traditional web usage mining, a session is not defined as an ordered set of pageviews, but as a set of ordered events. This comes from the realization that, all the actions of the user, including accessing and viewing a page, can be captured by events.

$$S = \{a_1, a_2, a_3, \ldots\}$$

Definition 3.5.4 (ismoregeneralthan, $\succcurlyeq$ relation) *We define the relation $isMoreGeneralThan(\succcurlyeq)$ over individuals, role assertions, atoms and atom-trees, which states that the left hand side is a more general term than the right hand side. Eq. 3.1 lists possible cases:* ■

$$
\begin{aligned}
&(1)\quad o_1 \succcurlyeq o_2 && \iff && C_1(o_1), C_2(o_2), C_1 \sqsupseteq C_2, o_1 \text{is anonymous} \\
&(2)\quad R_1(o_1, o_2) \succcurlyeq R_2(o_1, o_2) && \iff && R_1 \sqsupseteq R_2 \\
&(3)\quad R_1(o, o_1) \succcurlyeq R_2(o, o_2) && \iff && o_1 \succcurlyeq o_2, R_1 \sqsupseteq R_2 \\
&(4)\quad U_1(o_1, v_1) \succcurlyeq U_2(o_1, v_1) && \iff && U_1 \sqsupseteq U_2 \\
&(5)\quad \alpha_1 \succcurlyeq \alpha_2 && \iff && \alpha_1 \text{ and } \alpha_2 \text{ are of the same type, and either (1)-(4) holds} \\
&(6)\quad a_1 \succcurlyeq a_2 && \iff && a_1 = \{\alpha_{11}, \alpha_{12}, \ldots\}, a_2 = \{\alpha_{21}, \alpha_{22}, \ldots\}, \\
& && && \forall \alpha_{1i}, \exists \alpha_{2j}, \text{ s.t. } \alpha_{1i} \succcurlyeq \alpha_{2j}, \\
& && && \text{and } \alpha_{1i}, \alpha_{2j} \text{ are in the same level in the tree}
\end{aligned}
\tag{3.1}
$$

The semantics, under the interpretation $\mathcal{I}$ of ($\succcurlyeq$) should be clear from Equation 3.1. In plain English the operator $\succcurlyeq$, states that

1. An individual o_1, is more general than o_2, iff the class of o_1 is a superclass of the class of o_2, and o_1 is anonymous. If o_1 is a named individual, than it is not more general than another named individual.

2. An assertion $R_1(o_1, o_2)$ is more general that $R_2(o_1, o_2)$, iff their objects and subjects are equal and the first role is a super-role of the second.

3. An assertion $R_1(o, o_1)$ is more general that $R_2(o, o_2)$, iff they have the same object, and the subject of the first is more general than the subject of the second and first role is a super-role of the second.

4. A datatype assertion $U_1(o_1, v_1)$ is more general than $U_2(o_1, v_1)$, iff they have the same object and subject and U_1 is a super-role of U_2

5. An atom α_1 is more general than α_2 iff the atoms are both individual assertions, datatype property assertions, or object property assertions, and the first assertion is more general than the second according to rules 1–4.

6. An atom-tree a_1 is more general than a_2, iff for each atom in the first tree, there is a less general atom in the second tree in the same level.

Is more general than, $\succcurlyeq$, relation is *transitive*, *reflexive* and *antisymmetric* over the sets it is defined (i.e. the set of individual assertions, datatype assertions, role assertions and atom-trees). Equation 3.2 lists the properties in mathematical notation.

$$
\begin{aligned}
& a_1 \succcurlyeq a_2 \wedge a_2 \succcurlyeq a_3 && \Rightarrow \quad a_1 \succcurlyeq a_3 && \text{(Transitivity)} \\
& \forall a, \quad a \succcurlyeq a && && \text{(Reflexivity)} \\
& \forall a_i, a_j, \quad a_i \succcurlyeq a_j \wedge a_j \succcurlyeq a_i && \Rightarrow \quad a_i = a_j && \text{(AntiSymmetry)}
\end{aligned} \tag{3.2}
$$

With transitivity, reflexivity and antisymmetry, $\succcurlyeq$ defines a *partial order* over the sets it is defined. Nevertheless, $\succcurlyeq$ is not a total order, since some of the pairs are incomparable. In mathematical notation :

$$
\forall a_i, a_j, \quad \text{either } a_i \succcurlyeq a_j \mid a_i \succcurlyeq a_j \mid (a_i \not\succcurlyeq a_j \wedge a_j \not\succcurlyeq a_i) \tag{3.3}
$$

For notational convenience the relations, $isLessGeneralThan$, $\preccurlyeq$, and $strictIsMoreGeneralThan$, $\succ$, and $strictIsLessGeneralThan$, $\prec$, are defined as in Equation 3.4.

$$
\begin{aligned}
a_1 \succ a_2 &\iff a_1 \succcurlyeq a_2 \wedge a_1 \neq a_2 \\
a_1 \preccurlyeq a_2 &\iff a_2 \succcurlyeq a_1 \\
a_1 \prec a_2 &\iff a_1 \preccurlyeq a_2 \wedge a_1 \neq a_2
\end{aligned} \tag{3.4}
$$

To illustrate the $\succcurlyeq$ relation among atom-trees, some atom-trees are compared in Figure 3.5. The knowledge base contains concepts $C_1, C_2, \ldots$, Datatype roles $D_1, D_2, \ldots$, and object roles $O_1, O_2, \ldots$. C_1 is a super concept of C_2, D_1 is a super role of D_2, and O_1 is a super role of O_2.

The $\succcurlyeq$ relation is the key point of injecting semantics to the web usage mining process. It is defined over four sets, which are the set of individuals in the ABox, the set of datatype and object role assertions in the ABox, and the set of atom-trees. The relation leverages the class and role hierarchy semantic constructs of the terminological box, and usage of anonymous objects for representing the class. The domain and range restrictions for properties are used to limit the search space. Annotation properties of OWL is used in eliminating uninteresting patterns. Other constructs, other than the ones mentioned in this section are not used explicitly for the purposes of this work.

The frequent semantic event mining task is to find the frequent sequence of events, from the set of sessions. A frequent pattern, shows the sequence of events that the users perform frequently. Unlike, the sequence of URL's, the sequence of events is a more direct method to understand user's interaction with the site.

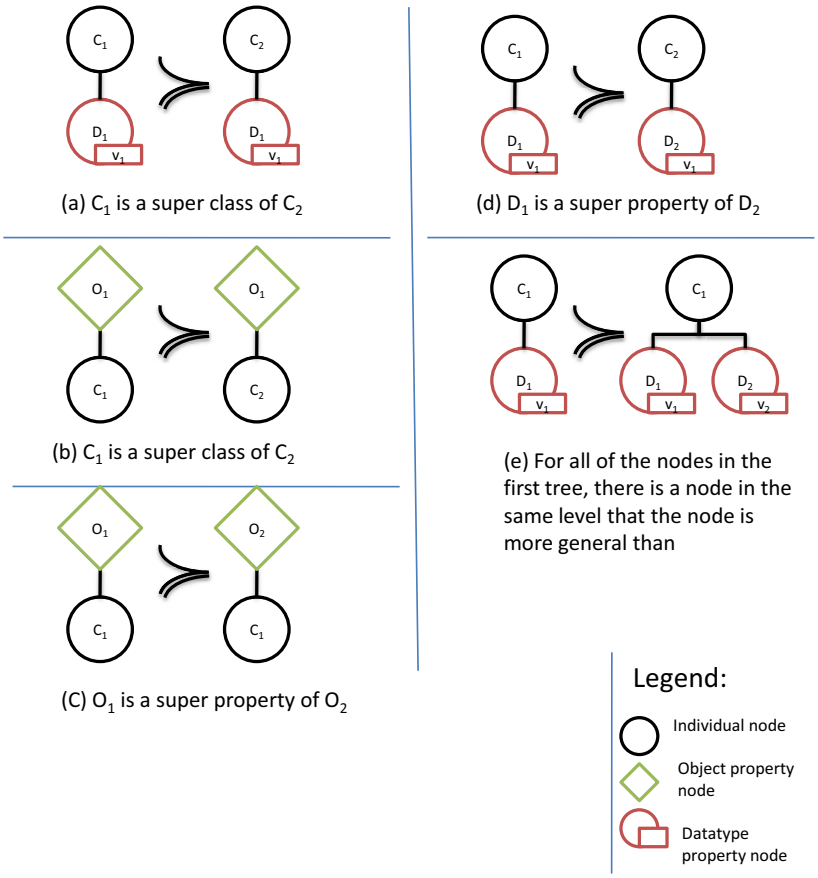

Figure 3.5: The $\succcurlyeq$ relation among atom-trees

Definition 3.5.5 (Pattern) *A pattern is an ordered set of atom-trees, representing a list of events.* ■

Patterns are similar to the sessions in the data set. However, the atom-trees in the patterns are constructed from the ones in the data set, using the $\succcurlyeq$ relation. For all the atom trees in the pattern, there exists an atom-tree in the data set, that the pattern is more general than. A pattern, Q, is denoted as:

$$Q = \{a_1, a_2, a_3, \ldots\}$$

A session S contains, or supports, the pattern Q iff Q is a subsequence of S, where the subsequence relation uses $\succcurlyeq$ for determining inclusion.

Definition 3.5.6 *Q is a subsequence of S iff for each element in Q, there is an element in S that is less general and its index is greater than the previous.* ■

$$
\begin{aligned}
&S =< a_1, a_2, a_3, \ldots, a_n > \\
&Q =< a'_1, a'_2, a'_3, \ldots, a'_m > \\
&\text{Q is a subsequence of S} \iff \forall i \in [1, m], \exists j_i \text{ s.t. } Q[i] \succcurlyeq S[j_i] \wedge j_i > j_{i-1}
\end{aligned} \tag{3.5}
$$

Definition 3.5.7 (Support of Pattern) *The support of a pattern is the number of sessions, that this pattern is a subsequence of.* ■

$$support(Q) = |\{ S \mid Q \text{ is a subsequence of } S \}| \tag{3.6}$$

where |.| gives the number of elements in the set.

The support of an atom-tree is similar:

Definition 3.5.8 (Support of atom-tree) *The support of an atom-tree is the support of the pattern with the atom-tree as its single element.* ■

The frequency is defined in terms of the support as

Definition 3.5.9 (Frequency) *The frequency of an atom-tree or pattern is the support of the atom-tree/pattern divided by the total number of sessions.* ■

$$frequency(Q) = \frac{support(Q)}{|D|} \tag{3.7}$$

Then for given Dataset, D, the problem is to find the set of patterns, $\mathbb{Q}$, with support greater than the threshold value, $minSupport$. In the next section an algorithm to search for this set is given.

$$\mathbb{Q} = \{Q \mid support(Q) \geq minsupport\} \tag{3.8}$$

3.5.2 Algorithm

The algorithm to find frequent sequences of atom-trees from the set of sessions is inspired by the level-wise search in the GSP algorithm. The algorithm consists of two phases. In the first phase the frequent atom-trees are found, and in the second phase, frequent atom-tree sequences are searched.

3.5.2.1 First Phase - Finding Frequent Atom-trees

In the first phase of the algorithm, frequent atom trees are found using the definitions in Section 3.5.1.2.

The algorithm is similar to the Apriori algorithm, in that, it iterates by generating candidates at each level, counting support of the candidates and generating new candidates from the previous ones.

A close examination of the definitions of support and $\succcurlyeq$ reveal that, there is an Apriori property among atom-trees. If an atom-tree is more general than the other, then its support is greater. Then by the following lemma, the algorithm searches for more general atom-trees and eliminates infrequent ones.

Lemma 3.5.10 *Given atom-trees* a_1, a_2, *if* $a_1 \succcurlyeq a_2 \Rightarrow support(a_1) \geq support(a_2)$

The proof for lemma 3.5.10 is straightforward from the definition of subsequence relation(Eq. 3.6) and support of pattern (Eq. 3.6). All the sessions supporting a_2 supports a_1 by the transitivity of $\succcurlyeq$.

To eliminate infrequent candidates in the early iterations, we begin counting the most general forms of the atom-trees and gradually refine the frequent ones. To obtain the initial set of atom-trees, a $getMostGeneralForms, \psi$, operation is defined over atom-trees.

Definition 3.5.11 **(**$getMostGeneralForms, \psi$**)** *The* $getMostGeneralForms, \psi$, *operation returns the set of trees, that are more general than the given atom-tree and that are not less general than any other tree in the set of atom-trees.* ■

$$\psi(a) = \{a' \mid a' \succcurlyeq a \wedge \nexists a'' \text{ s.t. } a'' \succcurlyeq a'\}$$

The pseudo code for the algorithm for finding most general forms of an atom-tree is given in Algorithm 3. The *getMostGeneralForms* function expects an AtomTree and returns a set of AtomTrees. The function expects the root node of the AtomTree to be a node containing an individual. Since all the events from the event tracking system is formed with an individual root, we can safely make this assumption. Top-level super classes of an ontology class are returned by the *getMostSuperClasses* method (not given). For each top-level super class of the root's OntClass, we construct a new tree with an anonymous individual atom of super class.

Algorithm 3 Finding Most General Forms of an Atom-tree

```
function getMostGeneralForms(a : AtomTree) : Set
  Ont ← Ontology given in OWL-DL
  F ← {}
  if a.root is an IndividualAtom then
    ontClass ← a.root.getOntologyClass()
    superClasses ← getMostSuperClasses(Ont, ontClass)
    for all superClass ∈ superClasses do
      atomTree.root ← IndividualAtom(superClass)
      F ← F ∪ atomTree
  else
    Signal error
  return F
```

Once the most general forms of the atom trees are found, the frequent trees are refined iteratively until no more candidates can be generated. A one-step refinement operator, ϕ is defined over the set of individual atoms, object and datatype property assertion atoms.

Definition 3.5.12 (one step refinement, ϕ) *Given individual atoms o_1, o_2, roles R_1, R_2, datatype roles U_1, U_2, the ϕ operator is defined over pairs where the first one more general than the other as :*

- *if* $o_1 \succcurlyeq o_2$, $\phi(o_1, o_2)$ *is the set of individual atoms that are individuals of direct subclasses of the class of* o_1*, and have the* o_2*'s class as a subclass.*

- *if* $R_1(o_1, o_2) \succcurlyeq R_2(o_1, o_2)$, $\phi(R_1(o_1, o_2), R_2(o_1, o_2))$ *is the set of role assertions, that have a direct subproperty of* R_1 *and have* R_2 *as a subproperty.*

- *if* $R_1(o, o_1) \succcurlyeq R_1(o, o_2)$, $\phi(R_1(o, o_1), R_1(o, o_2))$ *is the set of role assertions, that have the subject belonging to the set* $\phi(o_1, o_2)$

- *if* $U_1(o_1, v_1) \succcurlyeq U_2(o_1, v_1)$, $\phi(U_1(o_1, v_1), U_2(o_1, v_1))$ *is the set of datatype role assertions, that have a direct subproperty of* U_1 *and have* U_2 *as a subproperty.*

■

In mathematical terms the one-step refinement can be defined by:

$$
\begin{aligned}
&(1) \quad \phi(o_1, o_2) = \{o \mid o_1 \succ o \succcurlyeq o_2 \wedge \nexists o' \text{ s.t. } o_1 \succ o' \succ o\} \\
&(2) \quad \phi(R_1(o_1, o_2), R_2(o_1, o_2)) = \{R(o_1, o_2) \mid R_1 \sqsupset R \sqsupseteq R_2 \wedge \nexists R' \text{ s.t. } R_1 \sqsupset R' \sqsupset R\} \\
&(3) \quad \phi(R_1(o, o_1), R_1(o, o_2)), = \{R(o, o') \mid o' \in \phi(o_1, o_2)\} \\
&(4) \quad \phi(U_1(o_1, v_1), U_2(o_1, v_1)) = \{U(o_1, v_1) \mid U_1 \sqsupset U \sqsupseteq U_2 \wedge \nexists U' \text{ s.t. } U_1 \sqsupset U' \sqsupset U\}
\end{aligned}
\tag{3.9}
$$

The one-step refinement operator over the set of atom-trees is defined using the above definitions. We use the uppercase symbol, Φ for the operator.

Definition 3.5.13 *Given two atom-trees,* a_1, a_2*, and* $a_1 \succcurlyeq a_2$, $\Phi(a_1, a_2)$ *returns a set of atom-trees that is constructed by either*

- *refining a single node in the tree (by applying* ϕ*), or*

- *adding the mostGeneralForm of a child (by applying* ψ*) in* a_2*, that does not exists in* a_1*, to the corresponding parent node of* a_1*.*

■

Finally, Algorithm 4 contains the pseudo code for finding frequent atom-trees. The algorithm first finds the initial candidate set, and iterates until no more candidates can be generated. At

each iteration the atom-trees in the candidate set(C_i) is compared towards the dataset and the support of the candidates is incremented for each atom-tree in the dataset that is less general than the candidate. The next set of candidates is computed in this iteration by adding all the refinements of the candidate ($\Phi(a_c, a)$) with the atom-tree in the data set. Once all the candidates are counted, those with support at least *minSupport* are kept in L_i, as the frequent atom-trees. The candidates in C_{i+1} which are generated by refining some infrequent candidate are removed from the set.

Algorithm 4 Find frequent atom trees

function $generateC_1(D : DataSet) : C_1$

  $C_1 \leftarrow \{\}$

  for all session s ∈ D **do**

    for all atomTree a ∈ s **do**

      $C_1 \leftarrow C_1 \cup getMostGeneralForms(a)$

  return C_1

function $findFrequentAtomTrees(D : DataSet, minSupport) : L_1$

  $L_1 \leftarrow \{\}$

  $C_{2,3,...} \leftarrow \{\}$

  $C_1 \leftarrow generateC_1$(D)

  for (i = 1; $C_i \neq \emptyset$; i++) **do**

    for all session s ∈ D **do**

      for all atomTree a ∈ s **do**

        for all candidate atom-tree $a_c \in C_i$ **do**

          if $a_c \succcurlyeq a$ **then**

            a_c.count++

            $C_{i+1} \leftarrow C_{i+1} \cup \Phi(a_c, a)$

    for all candidate $a_c \in C_i$ **do**

      if a_c.count ≥ minSupport **then**

        $L_1 \leftarrow L_1 \cup a_c$

      else

        $C_{i+1} \leftarrow C_{i+1} \setminus \{a' \mid a' \in \Phi(a, a_c)$ for some a $\}$

  return L_1

The algorithm returns the set of frequent atom-trees. The set of frequent atom-trees is the types of events that occur frequently in the web site. These atom-trees and their supports are already interesting, since they capture usage statistics and how frequent events occur.

3.5.2.2 Second Phase - Finding Frequent Atom-tree Sequences

In the next stage of the algorithm the set of frequent atom-tree sequences are found. This stage of the algorithm is much similar to GSP [46]. In GSP, item hierarchies(taxonomies) can be introduced in the mining process. To mine frequent patterns with taxonomy, each data item is converted to an extended sequence, where each transaction contains all the items, and all the ancestors of the items in the hierarchy.

In the second phase of finding frequent event patterns, we use the $\succcurlyeq$ relation, which defines a partial order over the set of atom-trees. This order is used to introduce a taxonomy over the atom-trees, so that GSP is used with little modification.

For efficient comparison between atom-trees, we define and store a mapping from the atom-tree to its hash for each atom-tree in the set of frequent atom-trees, L_1. An important property of the the hash function is that it should not allow collisions, since once the dataset is converted, there is no way to compare the objects other than their hash values.

In the next step, each atom-tree in the data set is converted to a set of integers representing the atom-tree and it's ansectors according to $\succcurlyeq$ relation. For each atom-tree in the session, the set of frequent atom-trees that are more general than it is found, and the atom-tree is replaced by a set of hashes of the atom-tree and its parents. This operation converts a session into an ordered set of item-sets, where each item-set includes hashes of the atom-tree and its ancestors.

As an example for the subsequence relation, consider atom-trees $a_1, a_2, \ldots, a_{10}$, where $a_1 \succcurlyeq a_2 \succcurlyeq a_3$, $a_1 \succcurlyeq a_3$, $a_5 \succcurlyeq a_6 \succcurlyeq a_7$, and $a_8 \succcurlyeq a_9$. And further let $\forall_{i\in[0,10]}\ hash(a_i) = i$. The taxonomy for the atom-trees is given in Figure 3.6, where atom-trees are represented with their hashes. A sample input with 4 sessions is listed in Table 3.2. The atom-trees in the sessions are represented with their hashes. The second column contains the session events, while the third contains extended sessions. In the first row, the session with one event, $\langle 2 \rangle$, is augmented with 1, which is 2's parent ; i.e. $\langle \{1, 2\} \rangle$. In the next row, $\langle 7 \rangle$ is replaced with

$\{5, 6, 7\}$.

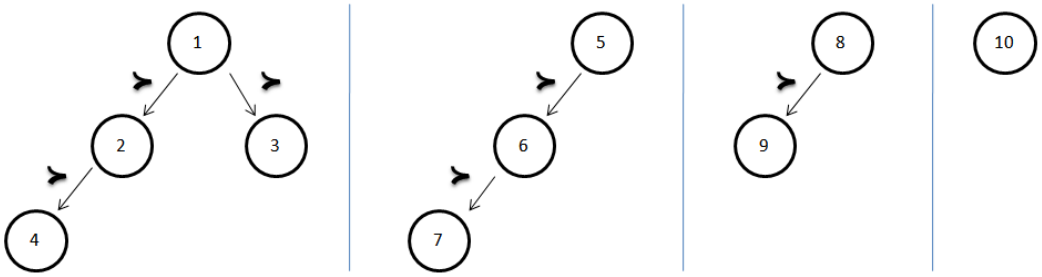

Figure 3.6: Example Atom-tree Hierarchy

Table 3.2: Sample Input Dataset for Second Phase

Session #	Session	Extended Session
1	$\langle 2 \rangle$	$\langle \{1, 2\} \rangle$
2	$\langle 7, 10, 8 \rangle$	$\langle \{5, 6, 7\}, \{10\}, \{8\} \rangle$
3	$\langle 3, 5 \rangle$	$\langle \{1, 3\}, \{4, 5\} \rangle$
4	$\langle 4, 9, 10 \rangle$	$\langle \{1, 2, 4\}, \{8, 9\}, \{10\} \rangle$

The subsequence relation between item-sets and sessions should be modified to include above semantics. Given a session, S, which is an ordered set of sets containing hash values of atom-trees, and a pattern ,Q, which is an ordered set of hash values of atom-trees the definition is as follows:

Definition 3.5.14 *Q is a subsequence of S, iff for each element q_i in Q, there is an element, $S[j]$ in S that contains q_i and its index is greater than the previous.*

■

More formally,

$$
\begin{aligned}
&S =< \{s_{11}, s_{12}, \ldots\}, \{s_{21}, \ldots\}, \ldots, \{s_{n1}, s_{n2}, \ldots\} > \\
&Q =< q_1, q_2, \ldots, q_m > \\
&\text{Q is a subsequence of S} \iff \forall i \in [1, m], \exists j_i \text{ s.t. } Q[i] \in S[j_i] \wedge j_i > j_{i-1}
\end{aligned}
\tag{3.10}
$$

With the $\succcurlyeq$ operator, and the modified subsequence definition, the task of finding frequent

event patterns is reduced to finding ordered frequent sequences in the dataset. We use an algorithm that is similar to GSP, however other sequential pattern mining algorithms such as SPADE [55] can be adapted in this phase. We have chosen and Apriori like algorithm since it offers straight-forward parallelization via MapReduce (Section 3.3.3).

The pseudo code for finding frequent atom-tree sequences is listed in Algorithm 5. The algorithm starts with initializing the candidate sets, $C_{2,3,...}$, and frequent sequence sets, $L_{2,3,...}$, to empty, and L_1 is initialized from phase 1. Note that, while the candidate sets, $C_{1,...}$ are different in phase 1 and 2, the set of frequent sequences, L_1 is the same. A mapping from atom-trees to their hashes is stored in variable μ. Then the dataset and patterns are converted to their hashes as discussed above. The algorithm, iterates over generate candidates, count candidates, and select candidates phases. New candidate set, C_{k+1} is generated by joining the frequent pattern set L_k with itself. Two sequences l_1 and l_2 of length k are joined if their first $k-1$ elements match, and the resulting sequence is generated by appending the last element of l_2 to l_1. If the candidate has any subsequence of length $k-1$ that is not frequent, then the candidate is filtered. In the counting candidates phase, for each the session, s, the set of candidates that are subsequences of the session is found and their support is incremented by 1. Hash-tree data structure or other optimizations in [46, 3] (such as eliminating sessions that does not contain any frequent item, etc.) can be used in this step. In the select phase, the candidate whose support is greater than the threshold $minSupport$ is selected as frequent. The algorithm then reconverts the pattern sets and returns the union of them.

3.5.3 Pattern Interestingness

In analyzing the resulting frequent patterns from the experimental setups (Chapter 4), a problem we faced was separating really interesting patterns from the obvious ones. In this process, we have identified two main problems that are causing subjectively uninteresting patterns.

The first problem is the atom-trees that lack some of it's children. By the definition of the $\succcurlyeq$ operator, an atom will or will not have all of the related properties. However such patterns are generated in internal steps in the algorithm and output as frequent patterns. As an example consider the `PlayVideoEvent`, `Video` classes and `video` object property relating the two. The atom-tree `PlayVideo` event but without associated video child is not interesting.

Algorithm 5 Find frequent event patterns

function $frequentEventPatterns(D : DataSet, minSupport)$

$L_{2,3,...} \leftarrow \{\}$

$C_{2,3,...} \leftarrow \{\}$

$L_1 \leftarrow findFrequentAtomTrees(D, minSupport)$(D)

$\mu \leftarrow getMapping(L_1)$

D $\leftarrow$ convertDataSet(D, μ)

$L_1 \leftarrow$ convertPatterns(L_1, μ)

for k = 2; $L_{k-1} \neq \emptyset$; k++ **do**

$C_k \leftarrow$ generateCandidates(L_{k-1})

countCandidates(D, C_k)

$L_k \leftarrow \{c \in C_k | c.count \geq minSupport\}$

$L_{2,3,...} \leftarrow reconvertPatterns(\mu, L_1, L_2, \ldots)$

return $L = \bigcup_k L_k$

function $countCandidates(D : DataSet, C_k)$

for all session s $\in$ D **do**

for all $c \in C_k$ s.t. c is a subsequence of s **do**

c.count++;

function $generateCandidates(L_k, minSupport) : C_{k+1}$

$C_{k+1} \leftarrow \{\}$

for all $l_1 \in L_k$ **do**

for all $l_2 \in L_k$ **do**

if $\forall_{j \in [1,(k-1)]}\ l_1[j] = l_2[j]$ **then**

$c \leftarrow \langle l_1[1], l_1[2], \ldots, l_1[k], l_2[k] \rangle$

if not $hasInfrequentSubset(L_k, c)$ **then**

$C_{k+1} \leftarrow C_{k+1} \cup c$

function $hasInfrequentSubset(L_k, c) : boolean$

for all subsequence s of length (k-1) of c **do**

if $s \notin L_k$ **then**

return TRUE

return FALSE

The second problem results from the fact that for some event patterns, the instances in that event may not be specific enough to be usable. For example, consider the classes `VisitContentEvent`, `Content`, `TopVideos`, and `Homepage` and object property `content` relating `VisitContentEvent` and `Content`. Further suppose that `TopVideos` and `HomePage` are subclasses of `Content`. With this ontology at hand, it is clear that the event (given in OWL abstract syntax, Section 2.3.2) :

```
Individual(type(VisitContentEvent) value(content Individual( type(Content)))
```

which is a pattern with an individual of type `VisitContentEvent` and single child `Content`, is not interesting at all. For the pattern to be interesting we need to refine the `Content` instance with either an instance of `HomePage` or `TopVideos`.

For filtering subjectively uninteresting patterns, we use predefined annotations. The developer of the ontology can optionally use two named boolean annotations, `is_required` and `need_refinement`, to add annotations to class or property definitions. The elimination of uninteresting patterns occur between the two phases of the algorithm. Unfiltered atom-trees from the first phase, is reported to the user, since the count of them is still interesting, but they are not used in the second phase.

An atom-tree is inspected for interestingness recursively starting from the root node. When we are at a node that contains an individual, we check the individual's class in the ontology to have the `need_refinement` annotation valued true. If so, we do not pass this atom-tree to the second phase. Moreover, we get the list of properties having the individual class as domain, and having `is_required` annotation as true. Then, we check each such property to be present among the children, and if not found we filter the atom-tree.

In this context, the interestingness of the patterns are defined subjectively. The ontology engineer is responsible for defining which classes or properties are too generic or necessary. In the following chapter, more concrete examples are given as a part of the experiments.

CHAPTER 4

EXPERIMENTAL RESULTS

4.1 Experimental Setup

For testing the usability and performance of the proposed algorithm and framework for discovering frequent semantic events, we have deployed two experimental setups. The first is a small site for listening streaming music and the second one is a web site for a major mobile network operator in Turkey. For both of the sites, the ontologies capturing domain events are developed and used during tests.

4.2 Music Streaming Site

4.2.1 Features of the site

The first experiments were performed in the music streaming site, called BulDinle [17]. The site offers searching and listening to songs that are uploaded to various online video sites. The site is targeted to the young population in Turkey, with an HTML/Ajax interface in Turkish. A screenshot of the BulDinle is given in Appendix A. The site offers a limited set of features and rather focussed on ease of use and speed. The features of the site is listed below:

- Make keyword searches: the results for the search are the list of songs that contain the query keywords in their title or metadata.
- Listen a song
- Add a song to the playlist

- Remove a song from playlist
- Play/Pause/Stop the song
- Go next/previous song in the playlist
- Send a song to a friend: send a mail to a friend with a URL to listen to the song.

The site enables to search over 7 million songs which are contained in other sites. The site sees a moderate amount of traffic with average 2700 visits and 8000 pageviews per day. Daily visit (session) counts, and pageviews between June 01, 2009 and June 29, 2009 are given in Figure 4.1 and Figure 4.2 respectively.

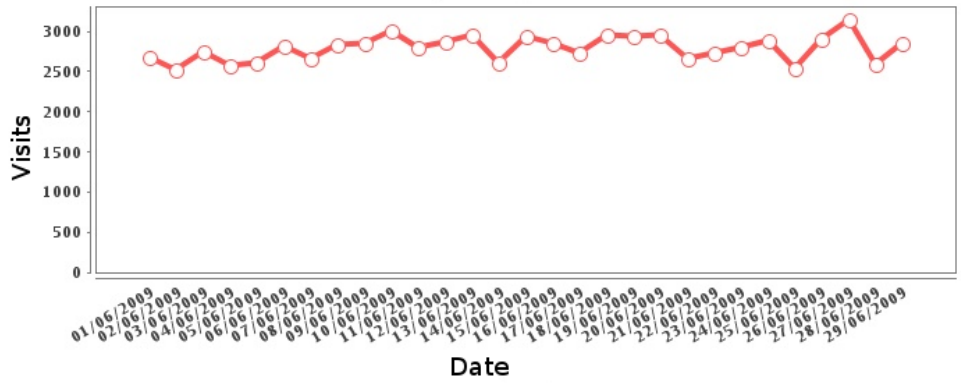

Figure 4.1: Daily Visit Counts for BulDinle

The user interface of the site is based on asynchronous communication, and all the user interaction with the site curiously occurs in one page. All of the URL's in the site starts with `"/index.php"`, which makes event tracking even more meaningful. Table 4.1 gives some frequently visited URL's of the site along with number of pageviews for each URL. As can be seen from the table, raw URL's are far from being interesting or useful.

4.2.2 Events in the site

For the music streaming site, we have defined several events that capture the user's interaction with the site. Furthermore for modeling the events in the site, we have developed an ontology

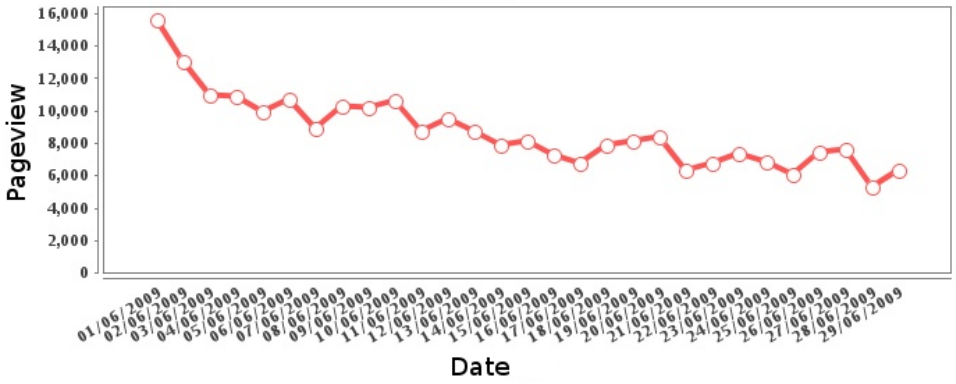

Figure 4.2: Daily Pageview Counts for BulDinle

Table 4.1: Sample URL's for BulDinle

URL	Num. Pageview
/	71710
/index.php?a=3__x8v4jz&k=6087559	5410
/index.php?a=3__x8g0vi&k=5508310	4698
/index.php?a=3__x71p76&k=3838533	2011
/index.php?a=3__x9bkpo&k=7064121	1851
/index.php?a=1__svgcxredkme&k=4851048	1672
/index.php?i=0&a=3__x8v4jz&k=6087559	1247

containing only the TBox of the ontology. The ontology, although simple, and the defined types of events proved to be sufficient for semantic analysis.

Figure 4.3 contains a visualization of the class hierarchy of the ontology for the music streaming site, and Figure 4.4 contains domain and range relations of the object properties. The ontology contains 7 classes, 1 object property and 2 datatype properties. The classes are, `Song`, `PlaySongEvent`, `AddSongToPlaylistEvent`, `SearchEvent`, `RemoveSongFromPlaylistEvent`, `ShareSongEvent`, the object property is `song` with domain `SongEvent` and range `Song`, and the datatype properties are `keyword` with domain `SearchEvent`, and `name` with domain `Song`. The events that are tracked in the site and sample atom-trees for the types of events are listed in Table 4.2. Five type of events are tracked with associated properties. The first column describes the intended action and the second column shows an example atom-tree

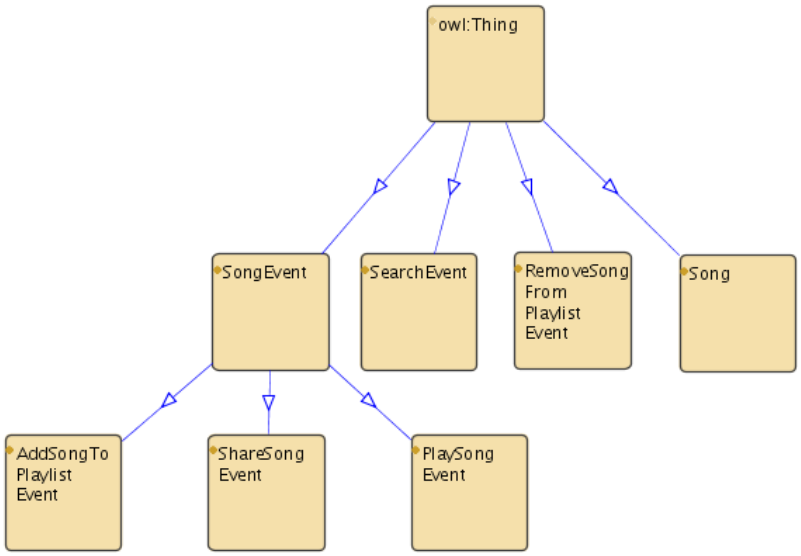

Figure 4.3: Class Hierarchy of the Ontology of Music Streaming Site

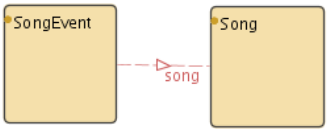

Figure 4.4: Domain-Range Relations of the Ontology of Music Streaming Site

resulting from the event. For example the second row of the table contains a sample event for playing a song. When a user performs this action, a new atom-tree is logged with the root node being an individual of class `PlaySongEvent`, and having an object property child, of type `song`. The `song` object property node has an individual atom child of type `Song`. The name of the song is a datatype property child of the song.

As mentioned in Section 3.5.3, not all the events that are counted as frequent is interesting. For example, in the above ontology, analysts might not be interested at all for events of `SongEvent`, rather that its one of it's subclasses, `PlaySongEvent`, `ShareSongEvent`, `AddSongToPlaylistEvent`. Similarly, a `PlaySongEvent` without an associated song might not be very informative in a pattern. Thus, to avoid uninteresting events occurring in patterns

Table 4.2: Events for BulDinle

Event	Sample atom-tree
Add a song to the playlist	`AddSongToPlaylistEvent(e1), song(e1,s1), Song(s1), name(s1, 'Billie Jean')`
Play a song	`PlaySongEvent(e2), song(e2,s2), Song(s2), name(s2, 'Thriller')`
Search Event	`SearchEvent(e3), keyword(e3, 'Michael Jackson')`
Remove a song from playlist	`RemoveSongFromPlaylistEvent(e4)`
Send a song to a friend	`ShareSongEvent(e5), song(e5,s5), Song(s5)`

of length more than one, we have defined `is_required` and `need_refinement` annotations as in Table 4.3. The `song` and `name` properties are defined to be required for their respective domains. However, the `keyword` property for domain `SearchEvent` is not required, since search events without drilling down to the keyword is considered to be sufficiently interesting. As can be realized, the determination of what constitutes an interesting event in a pattern, and making the annotations is in the responsibility of the domain analyst. The proposed system merely offers a way to help filter unsolicited results.

Table 4.3: Annotation Properties in the Ontology

OntResource	is_required	need_refinement
`SongEvent`		true
`song`	true	
`name`	true	
`keyword`	false	

4.2.3 Frequent semantic event patterns for the site

We used the event logs of the site for June 2009 to generate frequent event patterns. The logs contain 263076 pageviews and 284819 events in 75890 sessions, for an average of 3.47 pageviews and 3.75 events per session. 15959 sessions contains events, which gives an average number of 17.85 events per session. Relatively high number of events per session is suspected to be due to high engagement of core users with the site.

The algorithm was run on the described data set with $minSupport = 480$ ($frequency = 0.03$).

Table 4.4 contains the supports for most general atom-trees whose support is greater than *minSupport*. The atom-trees are given in OWL Abstract Syntax [53] in the first column and the supports are listed in second, the third column is the frequency of the event, and the last column states whether the event is interesting in terms of the definitions in Section 3.5.3.

Table 4.4: Event Patterns for Music Site, $\{\psi(a)\}$

Event	**Support**	**Freq.**	**Interesting**
`Individual( type(SearchEvent))`	6211	0.389	true
`Individual( type(RemoveSongFromPlaylistEvent))`	1500	0.093	true
`Individual( type(SongEvent))`	15240	0.954	false

The frequent events of most general form, denoted with $\{\psi(a)\}$, in Table 4.4 states that in 6211 of 15959 sessions, the user made a search, in 1500, the user removed a song from her playlist, and in 15240, the user made an action about a song, i.e. an event of one of the subclasses of `SongEvent`. In the next iteration of the algorithm, the frequencies for the one step refinements of the most general forms are computed. The resulting frequent events, denoted with $\{\phi(\psi(a))\}$ are listed in Table 4.5. The `SongEvent` in Table 4.4 is refined by adding a `song` property, the `SongEvent` is refined by subclasses `PlaySongEvent` and `AddSongToPlaylistEvent`. It is clear that, in 92.6% percent of the sessions, the user played a song, which confirms the expected user behavior.

Table 4.5: Event Frequencies for Music Site, $\{\phi(\psi(a))\}$

Event	**Support**	**Freq.**	**Interesting**
`Individual( type(SongEvent) value(song Individual( type(Song))))`	15240	0.954	false
`Individual( type(PlaySongEvent))`	14786	0.926	false
`Individual( type(AddSongToPlaylistEvent))`	4400	0.275	false

In the next iteration, the frequent events are refined again to calculate more refined patterns, $\phi^2(\psi(a))$. A sample set of the results are given in Table 4.6. In this refinement level, the names (more precisely unique identifiers) of the songs are counted for `SongEvent`s. For the popular songs, such as, the ones with ID's `'3__x8v4jz'`, `'3__x8v4jz'`, ... the number of times each song is played or added to a playlist can be seen from Table 4.7, which contains

the last level of refined events.

Table 4.6: Event Frequencies for Music Site, $\{\phi^2(\psi(a))\}$

Event	Support	Freq.	Interesting
`Individual( type(SongEvent) value(song Individual( type(Song) value(name '3__x9bdi9'))))`	1333	0.083	false
`Individual( type(SongEvent) value(song Individual( type(Song) value(name '3__x8v4jz'))))`	2397	0.150	false
`Individual( type(SongEvent) value(song Individual( type(Song) value(name '3__x8ib4g'))))`	1334	0.083	false
...	...	...	...
`Individual( type(PlaySongEvent) value(song Individual( type(Song))))`	14786	0.926	false
`Individual( type(AddSongToPlaylistEvent) value(song Individual( type(Song))))`	4400	0.275	false

Table 4.7: Event Frequencies for Music Site, $\{\phi^3(\psi(a))\}$

Event	Support	Freq.	Interesting
`Individual( type(AddSongToPlaylistEvent) value(song Individual( type(Song) value(name '3__x9bkpo'))))`	492	0.030	true
`Individual( type(PlaySongEvent) value(song Individual( type(Song) value(name '3__x8v4jz'))))`	2287	0.143	true
`Individual( type(PlaySongEvent) value(song Individual( type(Song) value(name '3__x8qhzu') )))`	956	0.059	true
...	...	...	...

In the first phase, a total of 55 events are found to be frequent with *frequency* greater than 0.03. 26 of the events are found to be interesting according to the annotations in the ontology. Both interesting and not interesting patterns are reported as one-item patterns, however only interesting patterns are used to generate patterns of length 2 or more. In the second phase of the algorithm, the frequent sequences of the frequent events are searched and a total of 139 frequent patterns are found.

A sample of patterns of length more than 1 are listed in Table B.1 in Appendix B. The table contains some interesting patterns for consideration. For example, the pattern number 1, shows that in 673 sessions, a search is performed after playing the song with ID `'3__x8ib4g'`. Pattern with number 4, indicates that the song with ID `'3__x8v4jz'`, which is played 2287 times according to 1 event pattern, is played after 5 searches, in 697 sessions, thus in 30% of the time, the song may be searched up to 5 times. This result alone may point to some lack of metadata of the song, or relevancy issues in the search system of the site. Pattern with number 4, shows that sequential removal of songs from playlist is frequent, which is clearly due to a lack of 'clear playlist' button in the playlist interface. The pattern with number 7, assures that searching is performed frequently in the site.

Figure 4.5 contains the number of candidates that are counted and the number of frequent patterns for each iteration of the algorithm. The first 5 iterations are in the first phase, while remaining iterations belong to the second. Figure 4.6 offers a closer look on the number of patterns. The number of candidates in phase 1 are far greater than the number of the frequent patterns. This is due to the relatively large number of distinct events generated as candidates. For example, at the fourth iteration, most of the candidates consists of a `PlaySongEvent` with a specific song ID. Generating a candidate for every distinct song event in the logs, may seem an overkill, however this is the only way to find frequent patterns in the refinement level of individual songs. In this respect, a parallelization with more traditional web usage mining setting is to generate a candidate for every distinct URL (including the query part) such as `'/index.php?a=3__x8v4jz&k=6087559'` instead of for every URL path, such as `'/index.php'`.

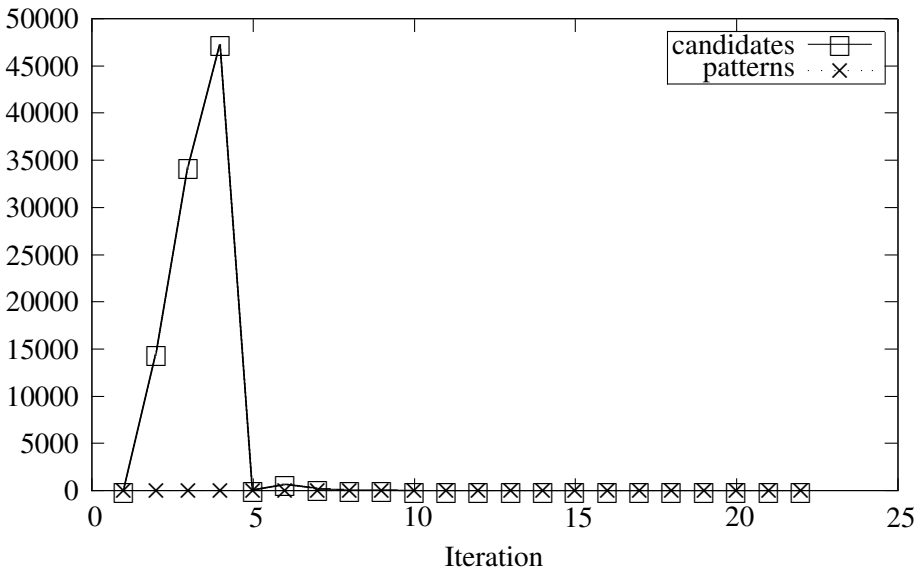

Figure 4.5: Number of Candidates and Patterns for Music Streaming Site

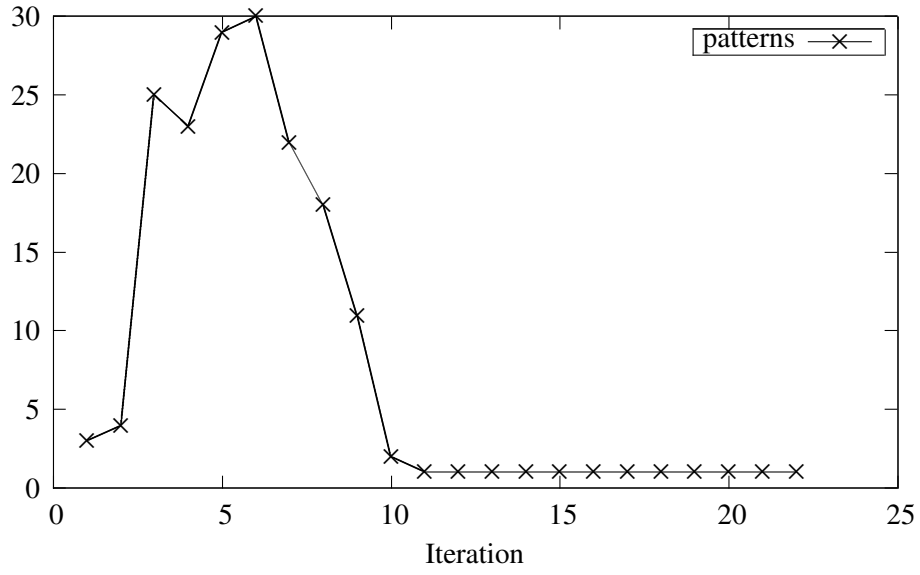

Figure 4.6: Number of Patterns for Music Streaming Site

4.3 Mobile Network Operator's Site

4.3.1 Features of the site

The second set of experiments was performed on the logs of the web servers of a large mobile network operator. The web site for the mobile network operator is a rather deep and complex one, with both static and dynamic content. The site enjoys a very large number of visitors.

Two days worth of logs, on 28-29 June 2008, as obtained from the company is used in the analysis. The number of pageviews and the number of sessions for each day is listed in Table 4.8. A total of more than 1 million pageviews occurred in 175K sessions.

Table 4.8: Pageview and Session Counts for Network Operator's Site

Date	# of Pageview	# of sessions
2008-06-28	568804	97114
2008-06-29	470415	78492

The web site is reported to be a core asset of the wireless network operator company, which enhances overall user experience, and allows online users to reach the features offered by the network. The web site includes a rich content with both static and dynamic pages. In the given time interval, 1,644 distinct URL paths (not containing query part of the URL), and 138,058 distinct URL's were accessed. The web site offers various set of features ranging from browsing static content to sending SMS online. A list of sample content and features of the site is given below.

- Browse through static content pages
- Perform a search in the site
- Download a document
- Use online services for account related actions
- Browse the content pages in English (default content is in Turkish)
- Seek help in help section

- Send an SMS online
- Buy prepaid subscriber units

Many more services, such as mobile services, specific sites for different types of paid plans, music streaming services, etc. can be reached from the main site. However, the log, as obtained from the company, contains only the logs for the main site, excluding most of the services that requires user sign-in. Due to privacy concerns, logs containing sensitive information was not sent over by the company for the analysis.

4.3.2 Preprocessing the logs

Unfortunately, we have not yet integrated event tracking of the Web Analysis project to the network operator's web site. Therefore, for this experiment, we have used the web server logs, as obtained from the company. The logs are formatted with the 'Combined Log Format with Cookie' [6] containing more than 10M lines of raw HTTP requests.

A preprocessing is performed over the logs to be used in the frequent event discovery phase. The logs are converted to event logs as tracked by the tracking module in the Web Analytics project. First, the logs are cleaned by removing non-page requests, and malformed lines. Only requests for HTML pages are kept, and all other requests, such as image, js, css files are removed from the log. A mapping is performed to convert page requests to events, which is detailed in Section 4.3.3. Finally the session and visitor identifiers are generated so that converted logs can be fed directly to the traffic analysis module.

Raw web server logs, did not contain a valid session identifier and a visitor identifier, so sessions had to be reconstructed reactively. For every distinct IP, we have generated a unique visitor ID, making the assumption that every distinct ip is a different visitor.

In constructing the sessions, we have grouped the pageview request by the visitor ID's and used time based heuristics to reconstruct the sessions. We have avoided more robust session reconstruction methods, such as using time and navigation oriented heuristics, or topology based heuristics [10], since they are considered orthogonal to the work. The frequent event pattern discovery system in production will always be used through the Web analytics project, specifically the log tracking module as discussed in Sections 2.1.3.7 and 3.4.

The preprocessing of the logs and defining path to event mappings is a one-time operation for the sake of experimentation. The web analysis project itself does not contain any apparatus for ontology creation, manipulation or semantic mapping. The project only allows the web sites to use event tracking by the Javascript API, and explore semantic event patterns. The creation of the domain ontology, uploading the ontology to the web analysis servers, and sending event tracking requests (according to the ontology) are the responsibilities of the web sites' administrators.

4.3.3 Events in the site

In this section the events in the site and the ontology is formalized. In the first subsection, types of events are defined and relevant detail about the site's ontology is provided. In the second, event mapping from the server logs are elaborated.

4.3.3.1 Site's Ontology

As discussed above, the logs contain data mostly about the static content. Thus the ontology and the events are defined according to the data at hand. In generating the ontology for capturing user interaction with the site, we have used both manual and automated methods.

We have defined 5 types of events, with classes `DownloadFileEvent`, `PageviewEvent`, `SearchEvent`, `VisitContentEvent`, and `SendSMSEvent` that can be performed on the site, which are listed in Table 4.9. The second column describes the intended action and the third column shows an example atom-tree resulting from the event. The first row indicates that whenever a page, which contains a known content in the ontology, is visited a `VisitContentEvent` is generated with a `content` property linking it with an object of the content's class. In this case the content's class is `Palm_treo_750`, indicating a content about the 'Palm Treo 750' mobile phone. The second row in the table, indicates that a pageview is performed but the content in the page is not associated with any class in the ontology. The URL of the page is linked to the event with `'url'` datatype property. The third action captures a search event together with the query, and the fourth action is for all types of downloads, where the `filename` is associated with the event. The last action is instantiated when a user sends an SMS from the web page.

Table 4.9: Events for the Network Operator's Site

#	Event	Sample atom-tree
1	Visit a page associated with a content	`VisitContentEvent(e1), content(e1,c1), Blackberry(c1)`
2	Visit a page not associated in the ontology	`PageviewEvent(e2), url(e2,'/c/oth/websmsframe.html')`
3	Make a search	`SearchEvent(e3), query(e3, 'blackberry')`
4	Download a file	`DownloadFileEvent(e4), filename(e4, '/downloads/windowsmobile6.pdf')`
5	Send an SMS	`SendSMSEvent(e5)`

The number of event types in the site may seem small, which is partly due to logs not containing mobile services, and online services. Also note that, some of the events, such as printing a page, cannot be reconstructed from the logs, since no pageview requests is performed for the event.Moreover, for some of the events (such as `SendSMSEvent`), the possible parameters for the event cannot be reconstructed. For some of these events, the web site tracks these statically by explicitly calling tracking methods in a competitor web analytic product. When the site switches to the web analysis product that is being developed here, a more extensive analysis of the site's events would become possible.

For the static pages, the site contains a site map, which includes the hierarchy, and a nice URL construction schema, from which the page content can be mapped. The part of the ontology about site's content was generated combining the site map, URL structure and manual effort. The sub-classes of the `Content` class, is highly specific to the knowledge domain, and extended know-how on the domain and knowledge of Turkish may be required for a complete interpretation. Since most of the content is in Turkish, some of the classes have Turkish names, however English translations of the class names are given in parenthesis in relevant context. Moreover the classes and their intended semantic interpretations will be detailed in Section 4.3.4 whenever necessary.

The terminological part of the ontology, contains 503 classes, 3 datatype properties and 1 object property. The datatype properties are `filename` with domain `DownloadFileEvent`, `query` with domain `SearchEvent` and `url` with domain `PageviewEvent`. The object property is `content` with domain `ViewContentEvent` and range `Content`.

Appendix D contains some detail about the ontology of the site. Figure D.1 gives a visualization of the top 2-level classes in the ontology. Some of the notable ones include, the 5 event types defined in Table 4.9, the `Content` class and `Anasayfa` (`HomePage` in English), `Bireysel` (`Personal`), `Kurumsal`(`Institutional`) and `Yardim` (`Help`) classes. The subclasses of the `Content` class capture the semantic content in the static pages. The ontology contains over 500 classes, and the deepest subclass in the hierarchy has a depth of 7, therefore only a portion of the ontology is given here. Figure D.2 contain 2-level subclasses of the `Bireysel`(`Personal`) class. Figure D.3 further details the subclass hierarchy of the `BireyselServislerMesajlasma` (`PersonalServicesMessaging`) class, which is itself a subclass of `Bireysel` (`Personal`). Lastly, in Figure D.4, the domain-range relation for the `content` object property is given. In accordance with the discussion in Section 3.5.3, not all the possible events are interesting. For eliminating uninteresting ones, we have used the `is_required` and `need_refinement` annotation properties in the ontology, which are listen in Table 4.10. The `Content` and it's direct subclasses are considered to be generic enough to be considered uninteresting, whereas indirect subclasses are accepted to provide sufficient detail. The `content`, `query` and `url` properties are needed for the event individuals in the domain to make sense. However, the `query` parameter is not considered essential for the `SearchEvent`.

Table 4.10: Annotation Properties in the Ontology

OntResource	**is_required**	**need_refinement**
`Content`		true
Direct sub-classes of `Content`		true
Indirect sub-classes of `Content`		false
`content`	true	
`filename`	true	
`query`	false	
`url`	true	

4.3.3.2 Event Mapping

As discussed above, the events for the site are not logged explicitly for the web site. However, according to the domain of the site, the events are defined and the logs are converted so that they contain the events as though generated by the event tracker. For each pageview, an event

is extracted and saved for later analysis.

For the mapping of pageview requests to event instances, we have developed a mapping schema. The URL of the requested page is analyzed to define the event type, and the event parameters. Table 4.11 overviews the mapping conditions and mapped events. Each pageview is first checked if it is a download event or search event. If not, the path of the URL is checked from the paths to contents table (detailed below) to see if the content in the page is a defined in the ontology. If so, an `VisitContentEvent` is generated, and the ontology class corresponding to the content is linked with the `content` property. If not, a `PageviewEvent` is generated as a last resort, linking the URL of the request with the `url` property

Table 4.11: Event Mapping for the Network Operator's Site

Condition	**Event**
URL path ends with one of 'pdf', 'zip', 'doc', ...	`DownloadFileEvent` with URL as `filename` property
URL path is '/yardim/arama' ('/help/search' in English)	`SearchEvent` with `query` property obtained from URL parameter 'q'
URL path is in path to Content mapping Table	`VisitContentEvent` with mapped content
None of the above matches	`PageviewEvent` with URL as `url` property

As mentioned in the previous section, the part of the ontology containing subclasses of Content are generated by a combination of the site map, the URL structure and manual effort. For most of the cases, the last part in the URL path is taken as the class name, and the directory structure in the path is converted to the class hierarchy in the domain. As an example, let's look at the `Palm_treo_750` class. The class hierarchy is constructed as follows: `Palm_treo_750` → `TurkcellPDA` → `MobileInternet` → `Services` → `Personal` → `Content`, where → indicates *is a subclass of* relation. The class names and class hierarchy are extracted from the URL, such as `/bireysel/servisler/mobilinternet/turkcellpda/palm_treo_750`. The mapping from URL's to classes is illustrated in Table 4.12.

The mapping is performed manually for the URL's, whose path names do not match the expected hierarchy. Generated classes were inspected, and verified to be descriptive of the content they represent.

Table 4.12: Event Mapping for the Network Operator's Site

	path	Ontology Class
English	/personal/services/mobileinternet/turkcellpda/palm_treo_750	`Palm_treo_750`
	/personal/services/mobileinternet/turkcellpda/	`TurkcellPDA`
	/personal/services/mobileinternet/	`MobileInternet`
	/personal/services/	`Services`
	/personal/	`Personal`
Turkish	/bireysel/servisler/mobilinternet/turkcellpda/palm_treo_750	`Palm_treo_750`
	/bireysel/servisler/mobilinternet/turkcellpda/	`TurkcellPDA`
	/bireysel/servisler/mobilinternet/	`MobilInternet`
	/bireysel/servisler/	`Servisler`
	/bireysel/	`Bireysel`

4.3.4 Frequent semantic event patterns for the site

We have run the proposed algorithm over the converted 2-day logs. Minimum support was chosen to be 5268, corresponding to 0.03 frequency. In determination of the minimum frequency to use, we have slowly decremented the frequency at each iteration until number of patterns is in the order of hundreds.

At the first phase of the algorithm, a total of 35 events are counted to be frequent. The most generic events are listed in Table 4.13 together with their support, frequency and whether they are interesting according to the interestingness criteria, defined in Section 3.5.3.

Table 4.13: Event Patterns for Mobile Network Operator's Site, $\{\psi(a)\}$

Event	Support	Freq.	Interesting
`Individual( type(SearchEvent))`	17171	0.098	true
`Individual( type(VisitContentEvent))`	171531	0.977	false
`Individual( type(PageviewEvent))`	67991	0.387	false
`Individual( type(SendSMSEvent))`	8716	0.050	true

Table 4.13 shows that nearly 10% of the sessions contain at least on search action, nearly all of the sessions contain browsing static content, and in 38% of the sessions, a page not categorized in the ontology is visited. Note that although `Individual( type(DownloadFileEvent))` was a candidate, it is not frequent, which indicates that downloading a file is not common.

The events in the above table, are refined in the next step of the algorithm. A sample of the resulting events are listed in Table 4.14. The events contain various pageview events with a specific URL, and a visit content event. For the SearchEvent, not a single query is identified as frequent.

Table 4.14: Event Patterns for Mobile Network Operator's Site, $\{\phi(\psi(a))\}$

Event	Support	Freq.	Interesting
Individual(type(PageviewEvent) value(url '/c/oth/galery.xml'))	7918	0.045	true
Individual(type(PageviewEvent) value(url '/c/oth/websmsframe.html?tab=dig_20080217_webmesaj'))	8385	0.047	true
Individual(type(VisitContentEvent) value(content Individual(type(Content))))	171531	0.976	true
...	...	...	...

At the next steps, the events are further refined as listed in Table 4.15, 4.16, 4.17, and 4.18 respectively. All of the events at these levels are patterns of `VisitContentEvent`s, with the `content` property of a subclass of `Content`. At each level, the `Content` classes are replaced by their subclasses. For example, the `Bireysel` (`Personal` in English) class in Table 4.15 is a super-class of the `BireyselServisler` (`PersonalServices` in English) class in Table 4.16, which is in turn a subclass of `Asistan` (`Assistant` in English) class in Table 4.17.

Table 4.15: Event Patterns for Mobile Network Operator's Site, $\{\phi^2(\psi(a))\}$

Event	Support	Freq.	Interesting
Individual(type(VisitContentEvent) value(content Individual(type(Yurtdisi))))	5826	0.033	false
Individual(type(VisitContentEvent) value(content Individual(type(Bireysel))))	79058	0.450	false
Individual(type(VisitContentEvent) value(content Individual(type(Anasayfa))))	128454	0.731	true
Individual(type(VisitContentEvent) value(content Individual(type(Yardim))))	7082	0.040	false
...	...	...	...

Table 4.16: Event Patterns for Mobile Network Operator's Site, $\{\phi^3(\psi(a))\}$

Event	Support	Freq.	Interesting
`Individual( type(VisitContentEvent) value(content Individual( type(BireyselTarifeler))))`	35328	0.201	true
`Individual( type(VisitContentEvent) value(content Individual( type(BireyselServisler))))`	30588	0.174	true
`Individual( type(VisitContentEvent) value(content Individual( type(Bireyselkampanyalar))))`	32968	0.188	true
...	...	...	...

Table 4.17: Event Patterns for Mobile Network Operator's Site, $\{\phi^4(\psi(a))\}$

Event	Support	Freq.	Interesting
`Individual( type(VisitContentEvent) value(content Individual( type(Asistan))))`	7881	0.045	true
`Individual( type(VisitContentEvent) value(content Individual( type(Ozelavantajlar))))`	18490	0.105	true
`Individual( type(VisitContentEvent) value(content Individual( type(Onlineislemmerkezi))))`	6268	0.036	true
`Individual( type(VisitContentEvent) value(content Individual( type(Hazirkart))))`	18033	0.103	true
...	...	...	...

Table 4.18: Event Patterns for Mobile Network Operator's Site, $\{\phi^5(\psi(a))\}$

Event	Support	Freq.	Interesting
`Individual( type(VisitContentEvent) value(content Individual( type(Hazirkarttarifeler))))`	10046	0.057	true
`Individual( type(VisitContentEvent) value(content Individual( type(Faturalitarifeler))))`	8140	0.046	true
...	...	...	...

In the first phase, a total of 35 events are found to be frequent with *frequency* greater than 0.03. 27 of the events are found to be interesting according to the annotations in the ontology. Both interesting and not interesting patterns are reported as one-item patterns, however only interesting patterns are used to generate patterns of length 2 or more. In the second phase of the algorithm, the frequent sequences of the frequent events are searched and a total of 173 frequent patterns are found.

Some of the subjectively interesting patterns with length more than 1 are listed in Table C.1 and C.2 in Appendix C. Found patterns are mostly of visit content events, with a few exceptions.

The pattern number 1 in the table indicates the frequency of sessions, in which the user first visited sub-categories of `BireyselKampanyalar` (`PersonalCampaigns` in English), and then `BireyselServices` (`PersonalServices` in English) Similarly, pattern 2 shows the support for patterns, in which the user visited `Anasayfa` (`HomePage` in English) and moved to `BireyselTarifeler` (`PersonalTariff`). Pattern 3 captures users visiting `BireyselServices` (`PersonalServices` in English), then making a search, whereas pattern 4 covers users making a search, then visiting `PersonalServices`. Other interesting patterns include the 6^{th} pattern, which captures the sessions browsing subsequently about `BireyselServisler` (`PersonalServices` in English), and the 7^{th} pattern which captures sessions browsing sub categories of `BireyselTarifeler` (`PersonalTariff`).

Lastly, the candidate counts and the pattern counts are plotted in Figure 4.7. First 7 iterations are for the first phase of the algorithm, while the remaining 7 belong to the second. As can be seen from the figure, the number of generated candidates can go as high as 12000, which is due to the large number of distinct URL's, and search queries. Note, that the number of candidates can easily be kept at the desired level, by allowing or disallowing more specific event constructs in the ontology. For example, if we do not include the `url` property in the `PageviewEvent`, then the number of candidates at this step is substantially reduced at the cost of ignoring URL-specific patterns. Figure 4.8 offers a closer look at the number of frequent patterns found at each iteration.

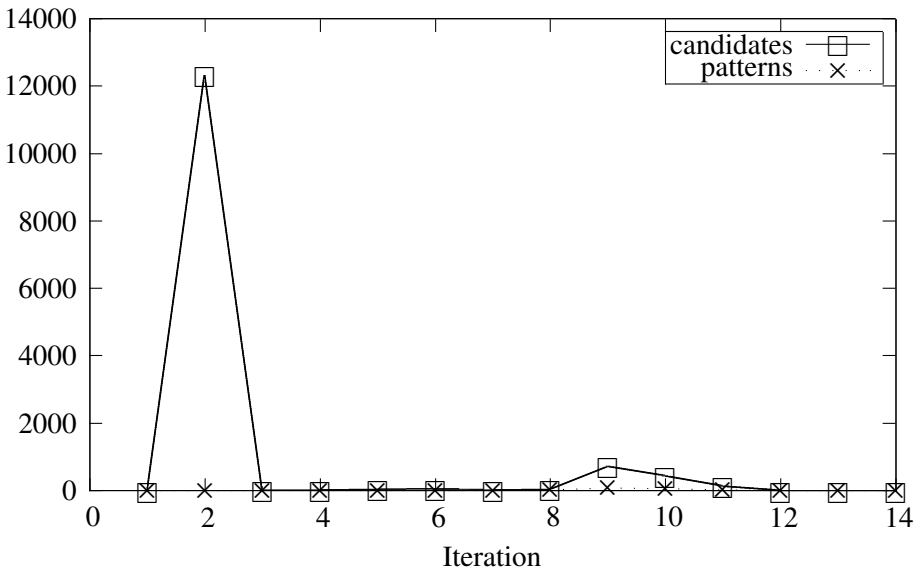

Figure 4.7: Number of Candidates and Patterns for Network Operator's Site

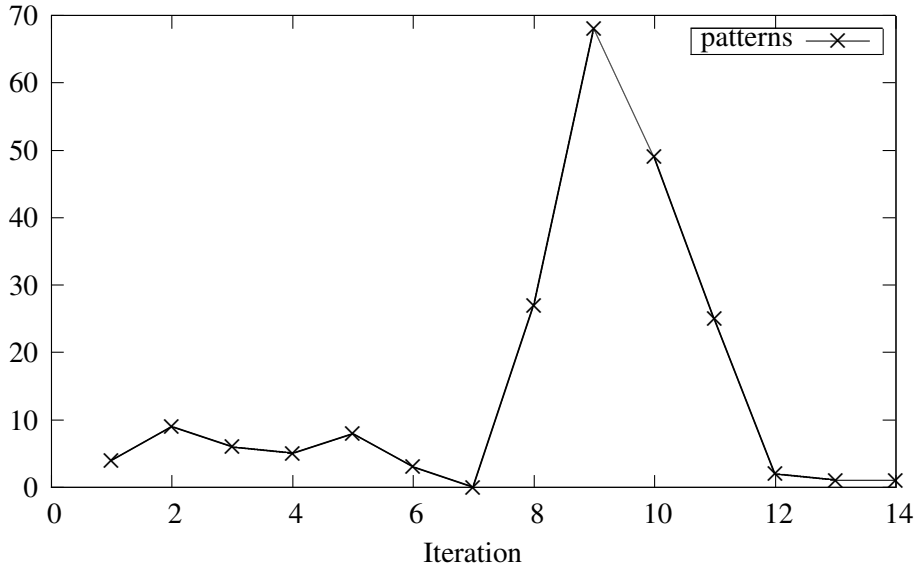

Figure 4.8: Number of Patterns for Network Operator's Site

CHAPTER 5

CONCLUSION

In this chapter, we give a comparison of the proposed system with earlier approaches and end with a brief conclusion.

5.1 Comparison with Earlier Approaches to Semantic WUM

The `VisitContentEvent` and `PageviewEvent` events defined in Section 4.3.3.1 deserves a special attention in the context of semantic web usage mining literature. In some of the earlier efforts, such as [13, 21, 16], a taxonomy of classes in the domain is developed, and every pageview is associated with one or more of the classes in the taxonomy. Then, instead of using raw URL logs, mining is performed from these content instances. However, as discussed in the Section 3.2, the pageview based user-access model is flawed in today's web.

In this sense our method of using semantic events to model arbitrary events (using `Visit ContentEvent` along with subclasses of `Content` for categorized content pages, using `Pageview Event` for uncategorized content, and other events for non-content actions) is more generic.

If the mapping from raw logs is performed using only instances of `PageviewEvent`, the approach is reduced to finding frequent URL patterns, similar to the traditional web usage mining setting. If the mapping is performed using only `VisitContentEvent` instances with a content taxonomy, the algorithm is reduced to the earlier methods. On the other hand, with the proposed system and with a complete domain model at hand, capturing, and counting complex usage behavior, such as 'Visit content about Cats', 'Search for kittens', 'Click on first link', 'Play a game about kittens, mice and dogs', becomes possible.

In [20], Dai et.al. cover the importance of the query strings (the part of the URL's after the '?' character), and states that in well designed sites, there is usually an explicitly available semantic mapping between query parameters and objects. For example, the query string `?action=viewitem&item=1234567&category=1234` from a fictional online book-seller web site, can indicate the viewing of the item 1234567 in the category 1234. However, typically the query parameter construction in real world web sites tend to be more complex than the given example. There may be lots of unrelated parameters (such as sessionid), input form values passed as URL parameters, misconfigured URL parameter names, etc. However, in our method, URL's and individual query parameters can be easily be tracked without an explicit site ontology as follows. Define the following classes in the ontology: `PageviewEvent`, `URL`, `Parameter`, object properties : `url`, `having`, and datatype properties `path`, `value`, and `name`. The `url` has the domain `PageviewEvent` and range `URL`. `path` has domain `URL` and `having` has domain `URL` and range `Parameter`. `name` and `value` properties have the `Parameter` as their domain. This short ontology basically captures the obvious : URL's have path, and a list of Parameters with name and value properties. If we use this ontology for capturing every pageview request, and use the semantic pattern mining algorithm, we can find patterns of URL paths, with or without specific URL parameters.

5.2 Conclusion

In this thesis, we have introduced semantic events and an algorithm for mining frequent semantic event sequences. The system has been tested on two real world web sites, and resulting patterns are interpreted in Chapter 4. We believe that proposed system has several advantages over traditional web usage mining systems and some of the earlier approaches to integrating semantics to the process.

First, the definition of events as objects with properties, and semantic events as mapped to objects in the ontology is a valid model for capturing web surfing behavior, as demonstrated in the experiments. It is discussed previously that pageview based access models are insufficient for capturing web site usage. The web analytics system, offers a way to capture user interaction through event objects and log them.

Second, the ordering relation among event atom trees is intuitive and sound. It is intuitive

because each of the 6 cases in Equation 3.1 makes sense according to their plain English definitions. The relation is sound because it depends on the model-theoretic semantics of the underlying logics. By using this relation, we can employ an ordering among possible events, and use the Apriori property to eliminate counting infrequent candidates.

Third, semantics is richly injected to the process. Much of the constructs in the knowledge representation system (OWL-DL in this case) is exploited in the algorithm. Previous efforts have mostly been focussed on class taxonomy, and limited support for properties. However, the proposed system, exploits classes, properties, sub-class and sub-property relations, individuals, property statements of individuals, and annotations. Moreover, possible future work for this project include taking into account other language constructs, such as enumeration, cardinality restrictions, grouping or aggregating datatype literals, etc. A curious extension would be using the property aggregator functions defined in [21], for grouping datatype literals.

It is clear from the examples of the music streaming site and the web site for mobile network operator that, the approach can be applicable to web sites with very different characteristics, usage patterns and site structures. The music streaming site is a one-page web site, with highly interactive and asynchronous access, while the mobile operator's site is filled with static content. Resulting patterns from these two web sites also show that, the patterns from the analysis are subjectively more meaningful than URL patterns. The patterns are also more easy to interpret by a site analyzer who may not be aware of the URL structure. We have seen that, without having a deep knowledge of the underlying technicalities of the sites, we can easily interpret the results of the algorithm.

Another advantage of the algorithm is that the Apriori-like architecture makes the algorithm parallelizable in a straightforward way. For the system to be used by hundreds of customers across many domains, with possibly millions of events and pageviews, it is vital that a distributed approach is employed. In the course of the project, the semantic mining algorithm is projected to be implemented over the MapReduce architecture (Section 3.3.3). We have already ported the AprioriAll algorithm so that the generate, count and select candidates phases are all run distributedly. The proposed algorithm can also be modified similarly.

REFERENCES

[1] S. Abiteboul, M. Preda, and G. Cobena. Adaptive on-line page importance computation. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 280–290, New York, NY, USA, 2003. ACM.

[2] R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. pages 207–216, 1993.

[3] R. Agrawal and R. Srikant. Mining Sequential Patterns. In *ICDE '95: Proceedings of the Eleventh International Conference on Data Engineering*, pages 3–14, Washington, DC, USA, 1995. IEEE Computer Society.

[4] G. Antoniou and F. vanHarmelen. *A Semantic Web Primer*. MIT Press, Cambridge, MA, USA, 2004.

[5] Apache Software Foundation. Hadoop Core Project. http://hadoop.apache.org/, last visited on August 2009.

[6] Apache Software Foundation. Log Files, Apache HTTP Server. http://httpd.apache.org/docs/2.0/logs.html, last visited on August 2009.

[7] F. Baader. Appendix: Description Logic Terminology. pages 485–495, 2003.

[8] F. Baader, I. Horrocks, and U. Sattler. Description Logics as Ontology Languages for the Semantic Web. In *Festschrift in honor of Jörg Siekmann, Lecture Notes in Artificial Intelligence*, pages 228–248. Springer-Verlag, 2003.

[9] F. Baader and W. Nutt. Basic Description Logics. pages 43–95, 2003.

[10] M. A. Bayir, I. H. Toroslu, and A. Cosar. A New Approach for Reactive Web Usage Data Processing. In *ICDE Workshops*, page 44, 2006.

[11] B. Berendt. Using Site Semantics to Analyze, Visualize, and Support Navigation. *Data Mining and Knowledge Discovery*, 6:37–59, 2002.

[12] B. Berendt, A. Hotho, and G. Stumme. Towards semantic web mining. In *International Semantic Web Conference (ISWC*, pages 264–278. Springer, 2002.

[13] B. Berendt, G. Stumme, and A. Hotho. *Data Mining: Next Generation Challenges and Future Directions*, chapter Usage Mining for and on the Semantic Web, pages 461–480. AAAI/MIT Press, 2004.

[14] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May, 2004.

[15] B. Bettina, A. Hotho, and G. Stumme. Towards Semantic Web Mining. In *First International Semantic Web Conference on The Semantic Web*, pages 264–278, London, UK, 2002. Springer-Verlag.

[16] A. Bose, K. Beemanapalli, J. Srivastava, and S. Sahar. *Advances in Web Mining and Web Usage Analysis*, volume 4811/2007 of *Lecture Notes in Computer Science*, chapter Incorporating Concept Hierarchies into Usage Mining Based Recommendations, pages 110–126. Springer Berlin / Heidelberg, 2007.

[17] Buldinle Web Site. http://www.buldinle.com/, last visited on August 2009.

[18] S. Chakrabarti. Data mining for hypertext: a tutorial survey. *SIGKDD Explor. Newsl.*, 1(2):1–11, 2000.

[19] R. Cooley, B. Mobasher, and J. Srivastava. Web Mining: Information and Pattern Discovery on the World Wide Web. In *ICTAI*, pages 558–567, 1997.

[20] H. Dai and B. Mobasher. Integrating Semantic Knowledge with Web Usage Mining for Personalization, 2005.

[21] H. K. Dai and B. Mobasher. Using Ontologies to Discover Domain-Level Web Usage Profiles. 2002.

[22] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, Berkeley, CA, USA, 2004. USENIX Association.

[23] P. Eric. *Web analytics demystified: a marketer's guide to understanding how your web site affects your business*. Celilo Group Media, 2004.

[24] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43, 2003.

[25] Google Analytics. http://www.google.com/analytics, last visited on August 2009.

[26] Google Analytics Event Tracking Guide. http://code.google.com/apis/analytics/docs/tracking/eventTrackerGuide.html, last visited on August 2009.

[27] J. Han and M. Kamber. *Data mining: concepts and techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.

[28] I. Horrocks, P. F. Patel-Schneider, and F. V. Harmelen. From SHIQ and RDF to OWL: The Making of a Web Ontology Language. *Journal of Web Semantics*, 1:2003, 2003.

[29] J. Józefowska, A. Lawrynowicz, and T. Lukaszewski. *Rules and Rule Markup Languages for the Semantic Web*, chapter Towards Discovery of Frequent Patterns in Description Logics with Rules, pages 84–97. Springer Berlin / Heidelberg, 2005.

[30] J. Józefowska, A. Lawrynowicz, and T. Lukaszewski. *Intelligent Information Processing and Web Mining*, chapter Faster Frequent Pattern Mining from the Semantic Web, pages 121–130. Springer Berlin / Heidelberg, 2006.

[31] A. Kaushik. *Web Analytics: An Hour a Day*. SYBEX Inc., Alameda, CA, USA, 2007.

[32] N. Khasawneh and C.-C. Chan. Active user-based and ontology-based web log data preprocessing for web usage mining. In *WI '06: Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 325–328, Washington, DC, USA, 2006. IEEE Computer Society.

[33] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, 1999.

[34] R. Kosala and H. Blockeel. Web mining research: a survey. *SIGKDD Explor. Newsl.*, 2(1):1–15, 2000.

[35] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tompkins, and E. Upfal. The Web as a graph. In *PODS '00: Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–10, New York, NY, USA, 2000. ACM.

[36] P. Leach, M. Mealling, and R. Salz. A Universally Unique IDentifier (UUID) URN Namespace. RFC 4122 (Proposed Standard), July 2005.

[37] A. Mikroyannidis and B. Theodoulidis. Heraclitus: A Framework for Semantic Web Adaptation. *IEEE Internet Computing*, 11(3):45–52, 2007.

[38] D. Nardi and R. J. Brachman. An Introduction to Description Logics. pages 1–40, 2003.

[39] D. Oberle, B. Berendt, A. Hotho, and J. Gonzalez. Conceptual User Tracking. In *AWIC*, pages 155–164, 2003.

[40] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[41] E. peng Lim and A. Sun. Web Mining - the Ontology Approach. In *International Advanced Digital Library Conference*, 2005.

[42] T. Robal and A. Kalja. Applying User Profile Ontology for Mining Web Site Adaptation Recommendations. In *Ioannidis, Y., Novikov, B. and Rachev, B. (eds) 11th East-European Conference on Advances in Databases and Information Systems (ADBIS 2007)*, 2007.

[43] S. Salın. Web Usage Mining and Recommendation with Semantic Information. Master's thesis, Middle East Technical University, 2009.

[44] M. Spiliopoulou, B. Mobasher, B. Berendt, and M. Nakagawa. A Framework for the Evaluation of Session Reconstruction Heuristics in Web-Usage Analysis. *INFORMS Journal on Computing*, 15(2):171–190, 2003.

[45] R. Srikant and R. Agrawal. Mining Generalized Association Rules. pages 407–419, 1995.

[46] R. Srikant and R. Agrawal. Mining Sequential Patterns: Generalizations and Performance Improvements. In *EDBT*, pages 3–17, 1996.

[47] J. Srivastava, R. Cooley, J. Srivastava, R. Cooley, M. Deshpande, and P. ning Tan. Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data, 2000.

[48] S. Staab, J. Angele, S. Decker, M. Erdmann, A. Hotho, A. Maedche, H.-P. Schnurr, R. Studer, and Y. Sure. Semantic community Web portals. *Comput. Netw.*, 33(1-6):473–491, 2000.

[49] G. Stumme and A. Hotho. Usage Mining for and on the Semantic Web. 2004.

[50] G. Stumme, A. Hotho, and B. Berendt. Semantic Web Mining: State of the art and future directions. *J. Web Sem.*, 4(2):124–143, 2006.

[51] W3C Recommendation. OWL Web Ontology Language Guide. http://www.w3.org/TR/owl-guide, last visited on August 2009, 2004.

[52] W3C Recommendation. OWL Web Ontology Language Overview. http://www.w3.org/TR/owl-features, last visited on August 2009, 2004.

[53] W3C Recommendation. OWL web ontology language semantics and abstract syntax. http://www.w3.org/TR/owl-semantics/, last visited on August 2009, 2004.

[54] T. White. *Hadoop: The Definitive Guide*. O'Reilly and Yahoo! Press, 2009.

[55] M. J. Zaki. SPADE: an efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1/2):31–60, 2001.

[56] Q. Zhao and S. S. Bhowmick. Sequential Pattern Mining: A Survey. Technical report, Center for Advanced Information Systems, School of Computer Engineering, Nanyang Technological University, Singapore, 2003.

APPENDIX A

A Sample Screenshot of Buldinle Music Site

Below is a sample screenshot of the music streaming site BulDinle [17].

Figure A.1: A Sample Screenshot of Buldinle Music Site

APPENDIX B

Event Patterns for Music Site

A sample of patterns of length more than 1, for the music streaming site [17], are listed in the following table.

Table B.1: Event Patterns for Music Site

Pattern #	Event Count	Event #	Pattern	Support	Frequency
...	...	...	...	...	...
1	2	1	Individual(type(PlaySongEvent) value(song Individual(type(Song) value(name '3__x8ib4g'))))	673	0.042
		2	Individual(type(SearchEvent))		
2	2	1	Individual(type(PlaySongEvent) value(song Individual(type(Song) value(name '3__x8v4jz'))))	480	0.030
		2	Individual(type(PlaySongEvent) value(song Individual(type(Song) value(name '3__x8v4jz'))))		
...	...	...	...	...	...
3	3	1	Individual(type(SearchEvent))	697	0.043
		2	Individual(type(RemoveSongFromPlaylistEvent))		
		3	Individual(type(SearchEvent))		
...	...	...	...	...	...
4	6	1	Individual(type(SearchEvent))	554	0.034
		2	Individual(type(SearchEvent))		
		3	Individual(type(SearchEvent))		
		4	Individual(type(SearchEvent))		
		5	Individual(type(SearchEvent))		
		6	Individual(type(PlaySongEvent) value(song Individual(type(Song) value(name '3__x8v4jz'))))		
5	6	1	Individual(type(RemoveSongFromPlaylistEvent))	502	0.031
		2	Individual(type(RemoveSongFromPlaylistEvent))		
		3	Individual(type(RemoveSongFromPlaylistEvent))		
		4	Individual(type(RemoveSongFromPlaylistEvent))		
		5	Individual(type(RemoveSongFromPlaylistEvent))		
		6	Individual(type(RemoveSongFromPlaylistEvent))		
...	...	...	...	...	...
6	7	1	Individual(type(SearchEvent))	1460	0.091
		2	Individual(type(SearchEvent))		
		3	Individual(type(SearchEvent))		
		4	Individual(type(SearchEvent))		
		5	Individual(type(SearchEvent))		
		6	Individual(type(SearchEvent))		
		7	Individual(type(SearchEvent))		
...	...	...	...	...	...

APPENDIX C

Event Patterns for Mobile Network Operator's Site

A sample of patterns of length more than 1, for the mobile network operator's site, are listed in Table C.1 and C.2.

Table C.1: Event Patterns for Mobile Network Operator's Site

Pattern #	Event Count	Event #	Pattern	Support	Frequency
...	...	...	...	...	...
1	2	1	Individual(type(VisitContentEvent) value(content Individual(type(Bireyselkampanyalar))))	5269	0.030
		2	Individual(type(VisitContentEvent) value(content Individual(type(BireyselServisler))))		
2	2	1	Individual(type(VisitContentEvent) value(content Individual(type(Anasayfa))))	16965	0.09
		2	Individual(type(VisitContentEvent) value(content Individual(type(BireyselTarifeler))))		
3	2	1	Individual(type(VisitContentEvent) value(content Individual(type(BireyselServisler))))	5378	0.031
		2	Individual(type(SearchEvent))		
4	2	1	Individual(type(SearchEvent))	5402	0.032
		2	Individual(type(VisitContentEvent) value(content Individual(type(BireyselServisler))))		
...	...	...	...	...	...
5	3	1	Individual(type(VisitContentEvent) value(content Individual(type(BireyselServisler))))	5778	0.033
		2	Individual(type(VisitContentEvent) value(content Individual(type(Anasayfa))))		
		3	Individual(type(VisitContentEvent) value(content Individual(type(BireyselServisler))))		
...	...	...	...	...	...
6	4	1	Individual(type(VisitContentEvent) value(content Individual(type(BireyselServisler))))	6327	0.037
		2	Individual(type(VisitContentEvent) value(content Individual(type(BireyselServisler))))		
		3	Individual(type(VisitContentEvent) value(content Individual(type(BireyselServisler))))		
		4	Individual(type(VisitContentEvent) value(content Individual(type(BireyselServisler))))		
...	...	...	...	...	...

Table C.2: Event Patterns for Mobile Network Operator's Site (Cont'd)

Pattern #	Event Count	Event #	Pattern	Support	Frequency
7	7	1	`Individual( type(VisitContentEvent) value(content Individual( type(BireyselTarifeler))))`	5280	0.031
		2	`Individual( type(VisitContentEvent) value(content Individual( type(BireyselTarifeler))))`		
		3	`Individual( type(VisitContentEvent) value(content Individual( type(BireyselTarifeler))))`		
		4	`Individual( type(VisitContentEvent) value(content Individual( type(BireyselTarifeler))))`		
		5	`Individual( type(VisitContentEvent) value(content Individual( type(BireyselTarifeler))))`		
		6	`Individual( type(VisitContentEvent) value(content Individual( type(BireyselTarifeler))))`		
		7	`Individual( type(VisitContentEvent) value(content Individual( type(BireyselTarifeler))))`		

APPENDIX D

Ontology Details for Mobile Network Operator's Site

In this section, several visualizations of the ontology of the mobile network operator's site are given. The figures are generated using the Protégé and Jambalaya tools. Please refer to Section 4.3.3.1 for a detailed discussion about the ontology constructs.

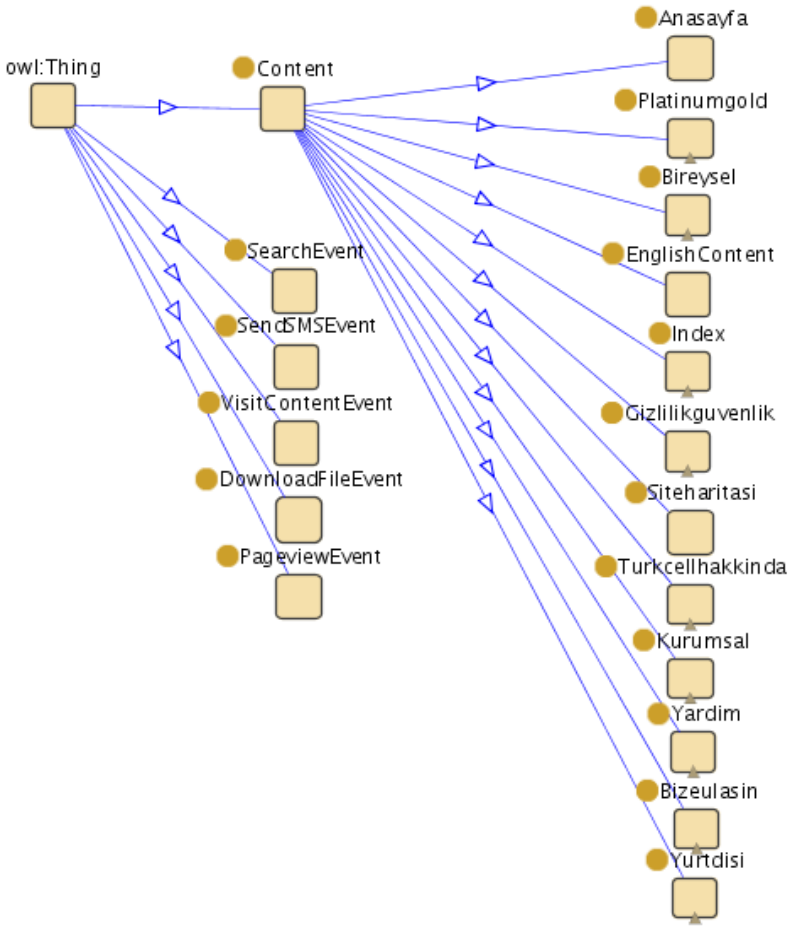

Figure D.1: Hierarchy of Top-Level Classes of the Ontology of Network Operator

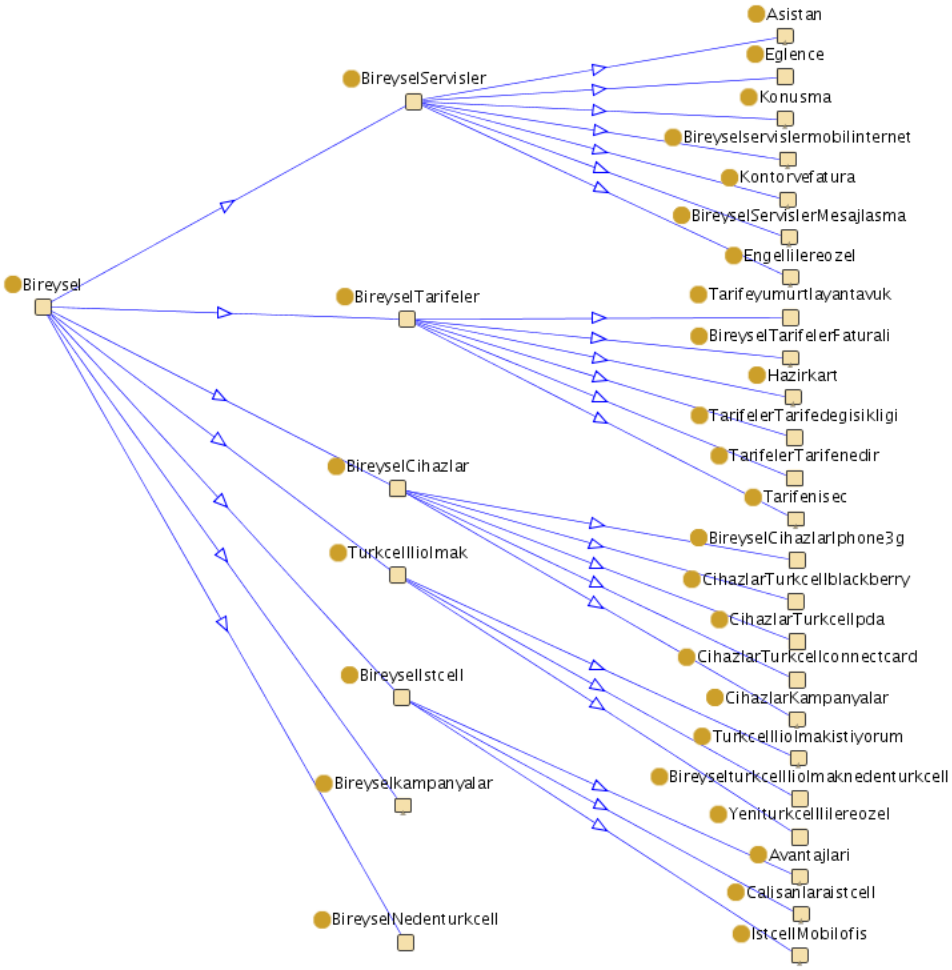

Figure D.2: Hierarchy for 2-level Subclasses of ’Personal’

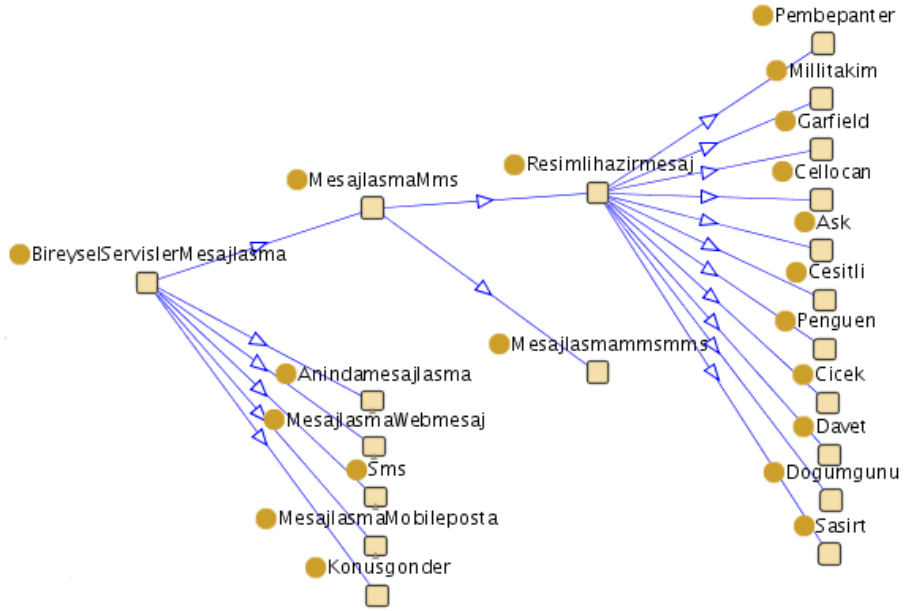

Figure D.3: Hierarchy for Subclasses of 'PersonalServicesMessaging'

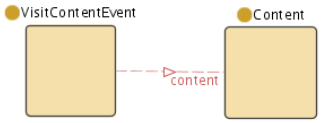

Figure D.4: Domain-range Relations of the Ontology of Network Operator