WebCaL
A Domain Specific Language for Web Caching

A Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Bachelor of Technology

by
Asha Tarachandani
Sumit Gulwani

under the guidance of
Dr. Deepak Gupta
Dr. Dheeraj Sanghi

to the

Department of Computer Science & Engineering
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

April, 2000

To our parents

Certificate

Certified that the work contained in the report entitled *WebCaL : A Domain Specific Language for Web Caching*, by Asha Tarachandani and Sumit Gulwani, has been carried out under our supervision and that this work has not been submitted elsewhere for a degree.

(Dr. Deepak Gupta) (Dr. Dheeraj Sanghi)

Abstract

Web Caching aims to improve the performance of the Internet in three ways - by improving client latency, alleviating network traffic and reducing server load. A web cache is basically a limited store of information which helps in presenting a faster web access atmosphere to the clients. The performance of a cache depends on proper management of this information and effective inter-cache communication. The existing web caches have simple and hard-coded policies which are not best suited for all environments. They offer limited flexibility and that too just in the form of changing some simple parameters such as cache size, peer caches etc. This drawback motivates the need for a framework for building new web caches tailored to specific environments. In this thesis, we describe a Domain Specific Language based on an Event-Action model using which new local web cache policies and inter-cache protocols can be easily specified.

Acknowledgements

We would like to express our gratitude to our supervisors Dr. Deepak Gupta and Dr. Dheeraj Sanghi for their guidance, motivation and invaluable suggestions at all stages of this project. It was their motivation more than anything else which always kept our spirits high. We would also like to thank them for their enthusiasm throughout the discussions we have had with them during the course of this project. Working with them was a real nice experience since, apart from enriching our academic knowledge, we also learnt effective time management and improved our presentation skills.

Our thanks are also due to Dr. Charles Consel, Dr. Gilles Muller and Luciano Porto Barreto for Sumit's valuable summer experience at Compose group, INRIA-IRISA labs, France.

Apart from our project supervisors, we would like to convey special thanks to Dr. R. Moona for his discussion on Automatic discovery of effective policies, Dr. S.K. Agrawal for his feedback on our framework for Event-Action systems and Dr. S. Ganguly for his encouragement to bring this up as a formal research work.

(Asha Tarachandani) (Sumit Gulwani)

Contents

Chapter 1

Introduction

Web Caching aims to improve the performance of the Internet in three ways - by improving client latency, alleviating network traffic and reducing server load. A web cache is basically a limited store of information which helps in presenting a faster web access atmosphere to the clients. A web caching infrastructure typically consists of a network of caches. Each cache temporarily stores web objects for later retrieval. If a request cannot be satisfied locally, the cache can also look for the corresponding web object in other caches before forwarding the request to the server [17]. In this way, the caches try their best to minimize the need to contact the server. If the request can be satisfied by a cache in the network itself, it not only improves client latency but also reduces server load and network traffic.

The internet is now being used by organizations with varying network conditions and to satisfy a variety of needs. The current web caches [1, 6, 16] lack the flexibility to adapt to these varied requirements. For example, a music-lover may like to cache audio files in preference to other types of files.Not only do the traditional caches lack in flexibility, the policies which they implement are not very effective since they are **uniform** for all types of documents and **fixed** for different network conditions at all times. The caches do not take into account the striking differences between various types of documents (text/html, text/plain, image/gif, image/jpg, etc.). For example, many of the web documents do not contain an expiry date in which case the cache has to make a decision whether or not to serve the local copy. This decision is generally based upon the age of the document. A cache should however also consider the fact that image documents do not expire as fast as the text documents while making this decision. Current web caches also do not take into account the changes in the environment. For example, when the internet load becomes very high, a cache should perhaps look for the document in the network of caches and serve it even without confirming its freshness from the server. Also, employing different inter-cache communication protocols upon change in network conditions may lead to better performance.

The naive and fixed policies followed by current caches are a great hindrance in achieving the full benefits of caching. This has led to research in the field of developing new policies. For instance, there have been attempts to find the optimal policy for removal of documents from the cache [18]. It was observed that the criteria used by the current proxy servers like LRU or LRU-MIN are among the worst performing criteria. New schemes like Greedy-Dual-Size algorithm [4], Latency Estimation Algorithm [19], Hybrid algorithm [19] have been proposed. Research is also being done in developing new inter-cache communication protocols [11, 12, 15]. The protocol currently used for this purpose is ICP, through which caches query each other for web documents. This simple protocol has not been much successful in improving performance of the caches. In the new protocols being developed, the basic idea is to let neighboring caches share content information. This will help in fast searches and accurate web query forwarding decisions. The employment of these new policies and protocols require either building a new cache server from scratch or modifying the code of an existing cache server neither of which is a desirable solution.

All these drawbacks motivate the need for a framework for specification of cache policies by users with varying needs. In this thesis, we propose a Domain Specific Language, WebCaL using which even dynamic and sophisticated policies for local web cache behavior and inter-cache communication can be easily specified. This thesis also makes a significant contribution to development of a generic framework for Domain Specific Languages which are based on Event-Action model. We have based the design of WebCaL on this framework thus rendering it extensible.

WebCaL has several advantages. It greatly reduces the time required to develop a new web cache. Building up a new cache and implementing it from scratch requires atleast two years of effort. WebCaL is aimed to help the user develop and use new cache policies and protocols very quickly thus reducing 2 years of effort to that of a few hours !! The syntax of the language is very natural and user-friendly. We believe it to be exhaustive for the domain of web caching. It is based upon a restricted version of C and SQL thus reducing the need to learn a new syntax. WebCaL can act as a test bed for new policies and protocols. It makes it possible to write a new policy or protocol, evaluate its performance and test it thoroughly using the complete program-execute-debug cycle very easily and quickly. WebCaL can also be used as a platform for proving certain properties of the Cache Protocol. For example, a Cache Protocol should be free of the cyclic effect i.e. if cache A contacts cache B for a document, and cache B further contacts cache C, then the cache C should not contact cache A or B. WebCaL distinguishes the role of three levels of users (viz. cache protocol writer, cache administrator and end-users) involved in web cache management.

CacheL [3] is also a Domain Specific Language for defining customizable caching policies but it does not capture the completeness of web caching domain. WebCaL is not only significantly more powerful but also has a strong theoretical foundation. Chapter 6 talks of the differences between WebCaL and CacheL. The rest of the thesis is organized as follows : in Chapter 2 we explain the basic concepts of a Domain Specific Language, Chapter 3 deals with description of an Event-Action model, Chapter 4 describes the actual language WebCaL, Chapter 5 gives an overview of the implementation of WebCaL. Further work has been suggested in Chapter 7.

Chapter 2

Domain Specific Languages

Domain Specific languages (DSLs) [7] are programming languages that are dedicated to specific application domains. They are less comprehensive than general-purpose languages like C or Java, but much more expressive in their domain. Such languages offer two main advantages:

1. **Productivity**: programming, maintenance and evolution are much easier (in some cases, development time can be ten times faster); re-use is systematized. For example, SQL is a DSL developed for fast and easy database management.

2. **Verification**: it becomes possible or much easier to automate formal proofs of critical properties of the software: security, safety, real time, etc. For example, a program written in a language with no loops is guaranteed to terminate.

WebCaL is also a DSL for web caching which offers several advantages.

2.1 Event Action Model

Many domains can be best viewed as based on based on Event-Action model. An event action model is a framework in which events occurring in the environment trigger actions [10, 14] . The triggered actions may generate more events that may trigger further actions. Events can be temporal (e.g. Timeout events) or non-temporal (e.g. change in value of a variable). Actions are high level features useful for the particular application domain. WebCaL is also based on an Event-Action model. We propose a generic framework for Domain Specific Languages which are based on Event-Action model. WebCaL is based upon this framework which is described below.

2.2 A novel approach for Event-Action Systems

Here we describe a general purpose abstraction for Event Action Model. A specific Event Action Model can be built upon this abstraction by defining events, actions and specifying the desired semantics.

We define a system as consisting of a set of finite state machines. A finite state machine (FSM) itself is a set of states. The states are of two types - public and private. Jumps are allowed from a state of a FSM to any state of the same FSM or to any public state of other FSMs. In case of a jump to a public state of an FSM, a return would bring the system back to the state from where the jump was made. A return from any state of an FSM would bring the system back to the state (of the predecessor FSM) from which the jump to a public state of this FSM was made. If the returning FSM has no predecessor, the system exits. Multi-jumps i.e. simultaneous jump to more than one state are also allowed. The system starts in the *start state* of the *start FSM* but as time progresses, it may simultaneously be in several states which may belong to different FSMs. This group of states is referred to as current-state-set. Each state has (1) Entry parameters, (2) Entry action (to be executed when a transition to this state is made) and (3) A set of pairs consisting of a *composite* event and the corresponding action. A system always listens for all the events specified in any of the states in the *current-state-set* and fires the corresponding action when any of those events occur.

A *Composite* event is a boolean expression of basic events. Every such composite event can be reduced to Disjunctive Normal Form - which is a disjunct of conjuncts and we refer to any conjunct of basic events as *Compound Event*. Thus, a state is equivalent to a set of compound events and corresponding actions where each compound event is a conjunct of events. For example, the composite_event-action pair E1&&(E2||E3) : A1 is equivalent to the following 2 compound_event-action pairs - (E1&&E2): A1 and (E1&&E3): A1

The occurrence of a compound event can have several semantics depending upon the type of events involved and the application. For example, let E1 be a simple Timeout event denoting the passage of every 10 seconds and E2 be a transition event with the predicate $X > 5$. A transition event involves a predicate and occurs at the instance when a predicate "becomes true" (not "is true"). In this scenario, we visualize several semantics that can possibly be attached to the occurrence of the compound event E1 && E2 depending on the application. For example,

1. The compound event is said to occur as soon as both the events occur atleast once.
2. The compound event is said to occur as soon as the transition occurs but after the timeout.
3. The compound event is said to occur as soon as the timeout occurs provided the transition has already occurred.
4. The compound event is said to occur as soon as the timeout occurs provided the transition has already occurred and the predicate $X > 5$ continues to be true at this instant.

If the compound event consists of three events, there can be even larger number of semantics associated with it.

We propose a simple scheme using which the desired semantics can be specified. This is done by treating events as objects with certain attributes and providing the option of defining a filter: a predicate involving these attributes. The compound event is said to occur if both the events occur atleast once and this filter is satisfied.

For example, every event has a time-of-occurrence attribute and the transition events have a current-value attribute which refers to the current value of the predicate. Thus, the above semantics can be supported with the help of the following predicates :

1. no predicate
2. E2.time-of-occurrence > E1.time-of-occurrence

3. E2.time-of-occurrence < E1.time-of-occurrence

4. (E2.time-of-occurrence < E1.time-of-occurrence) && (E2.current_value = true)

There is a special case of negation(!) of a time event provided it appears in conjunction with a set of events S. It denotes that the action should be fired when all the events in set S occur atleast once, the filter is satisfied but the time event does not occur. For example, !(timeout(10)) in conjunction with the set S would mean that the action should be performed when all the events in S occur atleast once *but within 10 minutes*. Note that negation of a time event, if it appears alone, makes no sense.

An action is made up of several statements. Each statement is a high level feature specific to the application domain. Three generic statements that exist in any Event-Action model are JUMP, RETURN and END. The semantics of these statements have been explained below.

A *path-stack* is defined as a stack of *stations* each of which represents a state of an FSM. If station S2 lies above S1 in the path-stack, it implies that there was a jump from the state represented by S1 to a public state of the FSM which contains the state represented by S2. Each station maintains as many *buckets* as there are events to be listened for in the state represented by that station. Each bucket corresponds to a particular event and holds the record of past occurrences of that event. The occurrence of an event can have several associated attributes such as time of occurrence, all of which are stored in the bucket.

Whenever all the buckets corresponding to the events in a compound event get non-empty and the filter is satisfied, the action associated with this compound event is triggered. All the buckets are popped. Several alternatives, each leading to a different semantic, are possible at this stage : (a) To clear all the buckets. This necessitates the occurrence of all the events in the compound event once again for the next triggering of the action (b) To leave the buckets unchanged. This keeps record of the fact that some events may have already occurred more than once before the action was triggered and counts these previous occurrences for the next triggering of the action. (c) To hand over the precise management of the buckets to the user.

Thus, the above mentioned system can be realized by maintaining a set of path-stacks. Note that the topmost elements of the path-stacks in this set constitute the current-state-set. Consider a path stack P. Let its topmost position represent a state S of FSM F. A **Jump** from S to n states of F and m (public) states of other FSMs is modeled by the following steps :

1. Make n copies of the path-stack P.
2. Replace the topmost position of these copies each with one of the n states.
3. Make m copies of the path-stack P.
4. Push the m public states one each on the m copies.
5. Destroy P.
6. Update current-state-stack.

Return from S is modeled by popping P. **End** in S is modeled by destroying P.

Chapter 3

Our perspective of Web Caching

Web Caching is a science in itself and should be studied in a proper scientific perspective. The physical component of a web cache is the available disk space to store data. And the soul consists of the various policies designed to effectively manage the data in this space and share the data with other caches so as to serve the purpose of providing a faster web-access atmosphere to the web users. The state of a web cache at any time is specified by the data currently stored in it and certain other variables (specifying its performance and current network conditions etc.). Our approach is to identify the type of data that can be stored in a web cache and the possible policies that can be applied to *manage* and *share* this data. We have provided a framework to easily handle all such possibilities. In this section, we focus on the type of data stored in the web cache.

3.1 The Web Cache Storehouse

The data stored in the disk space of a cache is critical in helping that cache as well as other caches to serve the client with fast and accurate information. This may contain three types of information:

1. Web Documents or Objects

2. Information associated with these web documents such as

    - Typical characteristics of the document. For example size, content-type, last-modified, expiry-date, links to other documents etc.

- The time when the document was brought in the cache.
- Access time - The time taken to access the document.
- Number of requests that have been made for the document since it was brought in the cache and by whom each request was made. Source of the document - it may even be a cache in the network.
- Client who requested the document - a cache when requesting web objects from another cache acts as a client for that cache.
- Some information associated with the request or the network environment at the time of the request. For example, network load at the time when request was made. This information may be used for proper estimation of average access time to a server or to some other cache.

3. Part of the information stored in other caches of the network.

We believe that the data that any cache would ever like to store will never fall outside the above domain of information. The above information can be organized in three types of tables :

1. **DOCUMENT_TABLE**: The main table containing the documents and other fields associated with these documents. The primary key of this table is the URL of the document. This table can automatically managed.
2. **PEER_TABLE(s)** : These tables contain information about other caches.
3. **HISTORY_TABLE(s)** : The users are given the flexibility to define their own tables and use them in any way they want. These tables can be used to maintain the relevant history of the cache. The statistics in these tables can be used to figure out any temporal or spatial correlations among the requests, responses, network environment etc. This information is used to improve the performance of the cache by various methods like prefetching all the spatially-correlated documents when a request for any one of them is made. We are working on a scheme for automatic discovery of spatial and temporal correlations. It is briefly described in Section 5.6

3.2 The Web Cache Variables

These include three types of variable :

- Performance measurement variables. e.g. Hit-Rate, Byte-Hit-Rate, Average access time of documents, etc.
- Current network environment variables. e.g. network load, state of peer caches etc.
- Inter-cache variables. e.g. Hit-Rate and Average access time of a cache with respect to. another cache

Chapter 4

WebCaL

WebCaL is a Domain Specific Language for writing Web Cache Protocols and Local Cache Customization. The purpose of WebCaL is to generate a full blown web cache server which will keep running in the background and perform certain actions in response to specific events. Hence WebCaL is based on an event-action model. In the domain of Web Caching, there are three types of basic events :

1. **Time events**: A Time Event matches the passage of a relative amount of time (e.g. every 30 minutes from now onwards) or the occurrence of an absolute time (e.g. 8 pm Sunday). Negation of Time Events is specified by the keywords "before" and "within". For example, within 2 hours matches only until 2 hours have elapsed, and before 8 am matches from now until 8 am.

2. **Transition events**: A Transition Event is represented as a boolean expression involving the variables and/or fields of database tables of the cache. It is satisfied when the expression "*becomes*" true (NOT "*is*" true). Hence the name Transition Event.

3. **Messages events**: A Message Event denotes the arrival of a message over the network.

An action is made up of several statements. In the domain of Web Caching, statements are high level features to manage the cache storehouse and its communication with other caches, clients or servers. WebCaL supports the following kinds of statements :

- **Statements for management of the cache storehouse**. The management may involve issues like removal of documents from the cache, updating the reference count of a document. This type of statements deal with all the basic database operations like insert, delete, update etc. Hence, the syntax is similar to SQL.

- **Statements for inter-cache communication**. The inter-cache communication will be used for serving a client with the requested document, requesting peer caches for a document etc. Only send statement falls in this group.

- **Specialized statements**. These are Place, Remove and Prefetch and these refer to placement, replacement and prefetching of web documents respectively.

- Other type of statements like definitions, assignment, jump, return, end, if-then-else.

4.1 Cache Policies - Refinements in the Event Action Model

A policy refers to a set of FSMs designed to perform a specific function of a web-cache. Each FSM is made up of several states and there are certain restrictions on the general event-action mapping that can be part of these states. The motivation behind classifying into several policies is to modularize the design such that each cache policy reflects a specific function of the cache which is fairly independent of other functions. The WebCaL policies are classified as:

1. **Global Policies** : These policies are specified keeping in view the whole cache network.

   (a) **Communication Policy** : This governs entire communication of a cache with other caches and web-servers. Communication with web-servers is done using HTTP protocol and that with other caches is done using pre-defined inter-cache communication protocols.

2. **Local Policies** : These policies are independent of the policies of the other caches. Any change in a local policy does not necessitate changes in any other policy.

   (a) **Placement policy**: This governs insertion of new documents into the database. For example, the placement policy can specify not to store any gif documents.

   (b) **Removal policy**: This governs the removal of documents from the database. It specifies when to remove the document and what documents to remove.

   (c) **Prefetching policy**: This governs the prefetching of documents from the web. For example a prefetching policy might specify to fetch the document with URL = "http://www.nyt.com" at 00:00 hours everyday. Placement, removal and prefetching policy together provide the complete background for management of web-documents in the cache and hence the automatic management of DOCUMENT_TABLE.

   (d) Local maintenance policy: This governs the management of HISTORY TABLES.

This classification highlights the different levels of cache management to be done by the corresponding level of users.

1. Local Cache Customization : This involves definition and application of the local policies. This is done by the end-users.

2. Inter Cache Communication Protocol : This involves definition and application of the global policies. The definition will be done by the cache protocol writer. While the application will be done by the cache administrator.

4.2 Placement Policy

This policy will typically be expressed as a uni-FSM and uni-state model. The only event allowed in this policy is arrival of a web document on the network. Since this event is implicit, the user need not mention it and this policy can simply be specified just by an action. The action should include a Store_If statement whose syntax is :

Store_If <Condition>
Example : The following statement directs the cache to store the document if the access time is more than 1 second.
Store_If *access_time>1 second*

4.3 Removal Policy

This policy will typically be expressed as a uni-FSM and uni-state model. The events allowed in this policy are the temporal events and those transition events whose predicate includes Current_Cache_Size. The action should include a Remove statement whose syntax is :

Remove
Where <SQL Predicate>
<**Restrictions**>

The *Where* clause selects the documents from the storehouse. The Restrictions come in three flavors :

1. **Till** <condition>
   The selected documents are removed in a random order till the condition is satisfied.

2. **Order By** <Attributes>
   Till <Condition>
   The selected documents are sorted in the specified order and are removed one by one in that order till the condition is satisfied.

3. **Till** <Condition restricting Current_Cache_Size> **Such_That** <Option> The selected documents are removed in a special fashion specified in the **Such_That**

clause till the condition is satisfied. The three options supported in the Such_That clause are *MIN_NUMBER_BYTES*, *MIN_NUMBER_DOCS* and *MIN_NUMBER_GROUPS*. The first option will typically be useful in improving the byte_hit_rate of the cache while others will be useful in improving the hit_rate of the cache.

Example :
Remove
Where content_type="gif"
Till Cache_Size = 0.5*(MAX_CACHE_SIZE)
Such_That MIN_NUMBER_BYTES

This example instructs the cache to remove the GIF documents till Cache is atmost half full such that the total number of bytes removed is minimal. Note that this is basically a form of Knapsack problem.

The removal policy should be designed carefully to prevent problems like cyclic removal and refetching of a document which may heavily degrade the performance of the cache.

4.4 Prefetching Policy

This policy will typically be expressed as a uni-FSM and uni-state model. All the three events i.e. Temporal events, Transition events and Message events are allowed in this policy. The action should include a Prefetch statement whose syntax is :

Prefetch <List of Terms>
A term can be a URL or an identifier denoting a list of URLs. Any term can also be followed by an integer within parenthesis. This integer denotes the depth upto which the recursive prefetching of documents will take place. Recursive prefetching of a HTTP URL means that the corresponding document will be prefetched and parsed followed by the prefetching of the files this document is referring to, down to the stated depth.

Note that prefetching does not necessarily mean that the documents will be prefetched directly from the server. Instead prefetching of documents will follow the routing policy (which is described below). Some examples of Prefetch policy are as follows :

Example : Prefetch the webpages www.yahoo.com, www.altavista.com, www.lycos.com whenever a request for www.yahoo.com of them is made.

Receive HTTP_Request from * − > If (request.URL = "www.yahoo.com")
Prefetch (www.yahoo.com, www.altavista.com, www.lycos.com)

Example : Prefetch the URL "www.timesofindia.com" recursively upto 3 levels at 00:00 hours if the network load is less than a certain threshold.

At 00:00::00 and (**network_load** < Threshold) − > **Prefetch** "www.timesofindia.com(3)".

4.5 Local Maintenance Policy

This policy will typically be expressed as a uni-FSM and uni-state model. All the three events i.e. Temporal events, Transition events and Message events are allowed in this policy. The action should include statements for management of HISTORY_TABLEs.

4.6 Communication Policy

This policy will typically be expressed as several FSMs, each consisting of several states. All the three events i.e. Temporal events, Transition events and Message events are allowed in this policy. The action should include statements for sending messages and storehouse update. This policy defines inter-cache communication protocols i.e. when, how and what do the caches exchange with each other. It serves two purposes :

- **Serving the client** : If the web document does not exist in the local cache, the cache contacts other caches or the origin server to fetch the document.

- **Exchange of a subset of storehouse information with other caches** : If the caches keep themselves up-to-date with the information about each other's storehouse, then they can be judicious in deciding whom to contact for a document. A cache stores the information about other caches in its PEER_TABLEs.

An example of this policy is provided in Appendix A (Section 8).

4.7 Automatic discovery of effective policies

Web Caching has a very bright future because the users do not aimlessly and randomly request the web pages. Several standard traces of web requests show that there is a significant correlation (both spatial and temporal) among the web objects based on their attributes. For example, the image documents of a very large size are generally never refetched thus indicating that they should be treated in a similar fashion with regard to placement and removal policies i.e. they should either not be cached or preferentially removed from the cache.

It may sometimes not be possible to figure out the effective policy. For example, it is difficult for the cache administrator of an organization to discover any correlations between the web requests made by a variety of users and accordingly code an effective placement policy. *Also the policies may be environment specific.* For example, there may be different temporal correlations of the requests made for web objects in the day time and those made during the night in the sense that it is more likely that an object fetched during the daytime will be fetched again within a short span of time as compared to an object fetched during night. This should have a direct implication on the removal policies and placement policies. *Even if some effective policy is figured out, it might not be very qualified in the sense that there may still be some scope for improvement.* For example, if it is figured out that image documents of small sizes are refetched frequently, then the removal policy would preferentially remove non-image documents or image documents of large sizes. It is however possible that the cache may have to throw away some of small image documents at some stage (because these are the only documents left in the cache and the cache needs to be freed more). In such a case, it would be prudent of the cache to use some other qualifying criterion in throwing away the documents e.g. access time. This motivates the need for automatic discovery of effective policies (removal, placement and prefetching) in certain scenarios [2]. We are still in the process of developing a technique for this purpose. Our technique is based on automatic management of HISTORY_TABLEs. The basic idea is similar to that used in maintaining branch prediction tables in a CPU. For each policy, one or several tables are maintained. Whenever, a decision corresponding to a particular policy is to be made, the global history bits for that policy are used to select a particular table. A table has several rows, each corresponding to a unique *category*. A hash function is used to map a document in question to a category. For each category of documents, certain state information is stored. A decision on a document is taken based on these state bits. Also, these state bits are modified based on the effect of that decision. Intuitively, this can be thought of as a very simple and cheap data mining technique.

4.8 Cache Initialization

This part deals with specification of

- Structure of the storehouse for each cache: This includes value of MAX_CACHE_SIZE and definition of various tables.

- Format and semantics of messages: Message are like C-structures supplemented with some refinements. Each type of message has a unique identifier. The user defines the fields of the message M and their types (and may also specify the default values for those fields). These default values may be constants. They may even be pointers to the storehouse information, cache variables or attributes of the message whose arrival on the network will trigger the action which contains a statement for sending the message M. Messages can also be inherited from other messages. A good reference for specification of messages can be found in [5].

Chapter 5

Implementation

We have developed a compiler, **wcc** (in C) which takes WebCaL specifications as input and generates the equivalent code in Java which runs in tandem with Event-Action backbone, Jigsaw and PostgreSQL to give a full-blown web-server.

Event-Action backbone is the Java code that implements our novel approach for Event-Action Systems as described in Section 2.2. **Jigsaw** is a Java-based Web Server which provides a complete HTTP 1.1 implementation. It has been professionally developed and is very well documented. The object oriented structure of the source code was extremely useful in extending it so as to suit our requirements. Of several web servers available, Jigsaw seemed to be the best choice as the backbone for implementation of WebCaL. **PostgreSQL** is a sophisticated Object-Relational DBMS, supporting almost all SQL constructs, including transactions, subselects, and user-defined functions and types. It is a highly efficient open-source database available currently. [13]

Chapter 6

Related Work

Reference [3] describes of a domain specific programming language called CacheL for defining customizable caching policies. Our work differs from theirs in the various aspects. Their programming language does consist of some high-level constructs but does not capture all aspects of web caching domain. For example, they cannot define new inter-cache communication protocols. They provide no support to find out the spatial and temporal correlations between documents. Our language does so through HISTORY_TABLEs which are either managed by the user or managed automatically. Their language utilizes default removal algorithms (e.g. LRU, LFU). Our language allows removal of documents based on any logical (domain related) criteria. They do not distinguish between different logical levels of users viz. cache protocol writer, cache administrator and end-users with respect to specification of policies. One of the most important ideas behind developing a Domain Specific Language is to be able to exploit the restrictions of the DSL to prove some properties of the application domain. Our language constructs allow easy and automatic verification of some critical properties of the web caching domain while CacheL lacks this concern. Our work is based upon a strong theoretical foundation for Event-Action systems. This foundation (which we have developed) has two main characteristics. First, it helps in easy development and maintenance of Event-Action systems. Second, it can be used to infer the power of that system. They do not talk of composition of events and their associated semantics.

Reference [14] summarizes the existing frameworks for development of Event-Action systems. Our work differs from all such frameworks in the many aspects. These frameworks assume a uni-FSM, uni-state model (according to our terminology above). In a general Event-Action system, not all events are to be listened to simultaneously. This can be realized only by modeling the system as consisting of several states. Also, modeling the system as a set of finite state machines (FSMs), where each FSM is made up of several states, not only results in a neat design but also renders extra power to the system. We have proposed a framework for development of Event-Action systems based on multi-FSM and multi-state model. Several semantics are possible while specification of the Event-Action based system. The existing frameworks have not realized these varied semantics. For example, the occurrence of composition of two events and maintenance of the history of events can have several semantics as described in Section 3. These frameworks just concentrate on event observation and notification. While we view event observation and notification as just a part of the 2-layered architecture that we have proposed for development of Event-Action based systems. With regard to this architecture, we foresee the development of a tool which will automatically generate the compiler for Event-Action based systems. This will take as input Event Specifications, Action Specification and Semantics specification and develop the Event-Action model for that.

Chapter 7

Conclusion and Further Work

With the expansion of WWW and the growing variety of web objects, the traditional policies employed by web caches are rapidly becoming inadequate. There is an increasing demand for specialized policies that can take into account the changes in the network environment and the striking differences between different types of documents. In this thesis report, we have proposed a Domain Specific Language (WebCaL) based on an Event-Action model using which a user can easily specify new local cache policies and inter-cache protocols.

This project can be extended in several ways. We here mention few of these ideas:

- *Development of a dedicated language for describing network protocols*: WebCaL offers some primitives for implementing a network protocols. However, we suggest the design and development of a new language dedicated to implementing only network protocols. It should have mechanism for automatic matching of requests and replies thus taking away some unnecessary burden from the user. It should also be able to provide a framework for proving some critical properties like absence of deadlocks, infinite loops etc.

- *Theoretical study of Event-Action Domain*: This work has proposed a new model of computation. Though we haven't studied this model intensively, we feel that it has power intermediate between Push Down Automata and Turing Machine. An exhaustive study of this domain in itself would be highly useful since it has immediate practical use. All Domain Specific Languages based on

Event Action framework can be modeled using this!

- *Automatic Discovery of Effective Policies*: Section 4.7 gives some insight into this possible idea. Infact, this was the point which appeared highly interesting to the Fifth Web Caching Workshop committee which reviewed our paper (and finally approved it! [8]) based on this project.

- *Measuring the performance of Web Caching policies described using WebCaL*: Currently, WebCaL supports few overall performance measurement parameters such as hit_rate, byte_hit_rate, average_access_time etc. More parameters can be included and a simulation platform can be provided to test the effectiveness of various cache policies. An extension and a new interface to WebCaL is required to support this.

- *Development of a dedicated language for precisely specifying the semantics of our novel Event-Action framework*: WebCaL assumes its own semantics suited for the domain of web caching. However, we suggest the design and development of a new language dedicated to implementing various possible semantics in an Event-Action framework. For example, it should expose mechanisms for bucket-management to the user.

Appendix A

Example

In WebCaL, each policy is written in a different file. Thus, a complete WebCaL program consists of five files, one each for *cache initialization policy, placement policy, removal policy, prefetching policy* and *communication policy.* These files are demarcated based on their extensions. This section provides an example of each of these policies.

A.1 Cache Initialization

The Cache Initialization Policy is written in a file with extension *.i.* The following is an example of this policy:

MAX_CACHE_SIZE = 200000;
DOC_INFO_TABLE = (**Content_Type**, **Size**, **Access_Time**, **Expiry_Date**);
@ *The user here chooses certain fields from some predefined fields*
@ *to be stored in the database*
CREATE TABLE PreciousList (URL varchar(50), Cache varchar(16));
@ *This is a user defined table*

@ *Message Definitions:*
MESSAGE MyInfo {
varchar[] URL = **SELECT** URL **FROM** **DOC_INFO_TABLE**
WHERE (**Access_Time** >= 10000);}

A.2 Placement Policy

The Placement Policy is written in a file with extension *.pl*. The following is an example of this policy:

@ The following statement directs the cache to store the document if the access time is more than 3 second.

Store_If Access_Time>3000;

A.3 Removal Policy

The Removal Policy is written in a file with extension *.r*. The following is an example of this policy:

```
#start Remove.Start
FSM Remove
   Start{
      ENTRY_CODE(
         EVENT Cache_Size>0.8*MAX_CACHE_SIZE Event1;
      )
      Event1 − > (
         Remove
         Where Content_type="gif"
         Till Cache_Size = 0.5*(MAX_CACHE_SIZE);
      )
   }
]
```

A.4 Prefetching Policy

The Prefetching Policy is written in a file with extension *.pr*. The following is an example of this policy:

```
#start Prefetch.Start
FSM Prefetch
  Start{
    ENTRY_CODE(
      EVENT AT 00:00:00 Event1;
    )
    Event1 -> (
      If Cache_Size<0.7*MAX_CACHE_SIZE{
        Prefetch "http://www.timesofindia.com";
      }
    )
  }
]
```

A.5 Communication Policy

The Communication Policy is written in a file with extension *.c*. An example of this policy is explained below followed by the how this would be written in a .com file.

There are three caches in the network namely C1, C2 and C3. They want to share with each other (every 10 hours) the list of the documents that had taken a long time i.e. atleast 10 seconds to access. This information is exploited by the caches when a request for a document is made by a client. In case of a cache miss, a cache checks the PreciousList table to figure out if any of its peer caches has the document. If none of its peer caches have advertised the document as a hotspot, it directly sends a request to the origin server. Else it sends a request for the document to the corresponding cache and waits for 5 seconds before it times out and forwards the request to the origin server. After making a request to the origin server, it again times out after 5 seconds in which case it sends NO DATA FOUND message to the client. If the peer cache or the server responds with the document, it forwards the same to the client. To start with, the web server will be in 2 states - the start state of SharingMyInfo FSM and the start state of Routing FSM.

```
#define HIT = ((Expiry_Date > Today) || (Expiry_Date == NULL))
#start SharingMyInfo.Start Routing.Start
FSM SharingMyInfo [
  State Start {
    ENTRY_CODE (
      EVENT EVERY(10 hours) Event1;
      EVENT Receive MyInfo Event2;
    )
@ The event-action mappings:
    Event1 -> (
      Send MyInfo to (C2, C3);
    )
    Event2 -> (
      INSERT INTO PreciousList VALUES (Event2.URL, Event2.sender);
    )
  }
]
@ The consistency policy for C2 and C3 is defined similarly.

FSM Routing[
  State Start {
    ENTRY_CODE (
     EVENT Receive HTTP_REQUEST Event1;
    )
    Event1 -> (
     String U = Event1.URL;
     String Sender = Event1.Sender;
     String Server = Event1.Server;
     date Expiry_Date;

     Expiry_Date = SELECT Expiry_Date from DOC_INFO_TABLE
     where URL = U;
```

```
    if (HIT) {
      HTTP_RESPONSE response = Form_Response_From_Table(U);
      Send response to Sender;
    }
    else {
      if (Sender in (C2, C3)) {
        Send HTTP_RESPONSE(-1) to Sender;
      }
      else {
        send HTTP_REQUEST(U) to (select cache from PreciousList
        where PreciousList.URL = U);
        Jump_To_State(Start, Waiting_Response_From_Cache(U, Server, Sender));
      }
    }
  )
}
State Waiting_Response_From_Cache(U String, Server String, Sender String){
  ENTRY_CODE (
   EVENT After(5 seconds) Event1;
   EVENT Receive HTTP_Response where (HTTP_Response.sender in (C2,C3)
   and HTTP_Response.URL== U) Event2;
  )
  Event1 -> (
   Send HTTP_Request(U) to Server;
   Jump_To_State(Waiting_Server);
  )
  Event2 -> (
   int Status_Code = Event2.Status_Code;
   if (Status_Code!=200) {
     Send HTTP_Request(URL) to Server;
     Jump_To_State(Waiting_Response_From_Server(Server, Sender));
```

```
      }
      else { HTTP_RESPONSE response = Form_Response_From_Event(Event2);
         Send response to Sender;
         END;
      }
    )
  }
  State Waiting_Response_From_Server(Server String, Sender String){
    ENTRY_CODE (
      EVENT After(5 seconds) Event1;
      EVENT Receive HTTP_Response where (HTTP_Response.sender == Server
      and HTTP_Response.URL == U) Event2;
    )
    Event1 − > (
      Send HTTP_RESPONSE(-1) to Sender;
      END;
    )
    Event2 − > (
      HTTP_RESPONSE response = Form_Response_From_Event(Event2);
      Send response to Sender;
      END;
    )
  }
]
```

Bibliography

[1] Apache Web Server http://www.apache.org/httpd.html

[2] Aubert O., Beugnard A. : Towards fine-grained adaptivity in web caches; In The fourth International Web Caching Workshop, 1999.

[3] Barnes J. and Pandey R., CacheL : Providing Dynamic and Customizable Caching Policies; In USENIX Second Symposium Internet Technologies and Systems [USITS99], October 1999.

[4] Cao Pei, Irani Sandy: Cost-Aware WW1 Proxy Caching Algorithms; In Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems, December 1997. Replacement Policies

[5] Chandra Satish and McCann Peter J. : Packet Types; in The Second Workshop on Compiler Support for Systems Software (WCSSS), May 1999. Messages

[6] Chankhunthod A., Danzig P.B., Neerdaels C., Schwartz M.F. and Worrell K.J. : A hierarchical internet object cache; In Proceedings of the USENIX 1996 Annual Technical Conference, 1996.

[7] Consel Charles and Marlet Renaud : Architecturing Software using a methodology for language development; In Proceedings of The Tenth International Symposium of Programming Languages, Implementations, Logics and Programs, September, 1998.

[8] Gulwani Sumit, Tarachandani Asha : WebCaL: A Domain Specific Language for Web Caching; to appear in the Fifth International Web Caching and Content Delivery Workshop, May 2000.

[9] Jigsaw- The W3C's Web Server: http://www.w3.org/Jigsaw/

[10] Krishnamurthy B. and Rosenblum D.S. : Yeast: A General Purpose Event-Action System; in IEEE Transactions on Software Engineering, Vol. 21, No. 10, October 1995.

[11] Lixia Zhang, Soctt Michel, Khoi Nguyen, Adam Rosenstein, Sally Floyd and Van Jacobson : Adaptive Web Caching: Towards a new caching architecture; In Third International WW1 Caching Workshop, June 1998.

[12] Malpani Radhika, Lorch Jacob, Berger David: Making World Wide Web Caching Servers Cooperate; In Fourth International WW1 Conference, December 1995.

[13] PostgreSQL http://www.postgresql.org/

[14] Rosenblum David S. and Wolf Alexander L. : A Design Framework for Internet-Scale Event Observation and Notification; In ACM SIGSOFT Fifth Symposium on the Foundation of Software Engineering, 1997.

[15] Rousskov A., Wessels D. : Cache Digests; In Proceedings of the third International Web Caching Workshop, 1998; http://ircache.nlanr.net/ wessels/Papers/ These papers talk that the caches should talk to each other

[16] Squid Web Proxy Cache, http://squid.nlanr.net

[17] Wessels D., Claffy K. : ICP and the Squid Web Cache; In IEEE Journal on Selected Areas in Communication, April 1998, Vol. 16, #3, pages 345-357; http://ircahe.nlanr.net/ wessels/Papers/

[18] Williams S., Abrams M., Standridge C.R., Fox E.A., Abdulla Ghaleb: Removal Policies in Network Caches for World-Wide Web Documents; In ACM SIGCOMM, 1996.

[19] Wooster R.P. and Abrams M. : Proxy Caching that estimates page load delays; In Computer Networks and ISDN Systems 29, 1997.