RETHINKING THE WEB STRUCTURE: FOCUSING ON EVENTS TO CREATE BETTER INFORMATION AND EXPERIENCE MANAGEMENT

A Thesis
Presented to
The Academic Faculty

by

Derik L. Pack

In Partial Fulfillment
of the Requirements for the Degree
Masters of Science

School of Electrical and Computer Engineering
Georgia Institute of Technology
July 2004

RETHINKING THE WEB STRUCTURE: FOCUSING ON EVENTS TO CREATE BETTER INFORMATION AND EXPERIENCE MANAGEMENT

Approved by:

Professor Jain, Advisor

Professor McClellan

Professor Wills

Date Approved: July 8, 2003

ACKNOWLEDGEMENTS

Both professionally and personally there are a lot of people who have helped me get this far. Professionally, I'd like to thank Ramesh and Rahul for giving me such an inspiring environment for research and intellectual growth. You both have always had your doors open no matter how busy you were. Personally, I'd like to thank my roommates who were always around to help me run amok when the time was right. I'd also like to thank Steve for helping me a little farther along on my spiritual journey to find a better balance between my scientific nature and the supernatural world. Lastly I'd like to thank mom and dad. You both taught me so many of the lessons I needed to get this far.

Thanks you everyone for the support you've given me.

Derik Pack

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The objective of the following research is to investigate the problem of information management and conveyed experience on the World Wide Web (WWW) when multimodal sensors and media are available. After studying related areas of work about the web and heterogeneous media, it became apparent that one of the main challenges of the area is the semantic unification of heterogeneous media. This thesis will introduce an event-based model to semantically unify media. An event is defined as something of significance that takes place at a given time and location. Using this definition and the corresponding model, a system will be designed to illustrate practical use cases for events.

CHAPTER 2

LITERATURE REVIEW

The proposed goal to semantically unify multimedia is relevant to several fields. These include human computer interaction research, World Wide Web development, and Experiential Computing Research.

2.1 World Wide Web Development

Program development for the World Wide Web began in 1990 after a publication by Tim Berners-Lee during the preceding year on information management using hypertext [3]. Lee championed the use of hypertext as a means to link related information because the links remove a fixed structure from the information [12]. Four years later, the WWW was first offered to home computer users. Over that time, the load on the first web server had increased over a thousand fold. With the expansion of the WWW, it was necessary to create an organization to facilitate inquiry and research into its continued development. The organization, the World Wide Web Consortium (W3C), came into existence on October 1st of 1994 [3].

2.1.1 Search Engines

The explosion of web pages resulted in a need to search for information. This need brings about the first generation of search engines. Search engines fall into four categories: human indexing, automatic indexing, agent based indexing, and meta-data indexing. Agent based indexing and automatic indexing overlap in many respects, and meta-data indexing can be done using either. Two of the early search engines, the World Wide Web Worm and Yahoo, indexed by automatic and human indexing respectively. Human indexing proves to be far more accurate than automatic indexing,

but automatic indexing proves to be far faster [20].

Early automatic indexing systems had a far lower correctness than human indexing systems because there was no way to judge the importance of content on a particular web page. With this in mind, Google's developers implement the page rank system [11]. This system adds a link based weighting system to word indexing. If similarly worded links to a given page are found on many websites, then searches for those words yield a higher probability for returning that particular website. Popular websites are also given a higher weight, and that higher weight increases the weight to websites linked to those popular sites.

Googles system of automatic indexing seemingly adds a peer-based rating by giving more importance to sites that a large numbers of pages linked too. This rating could be skewed through a process called Google Bombing. Google bombing takes place when a developer intentionally creates multiple websites with similar link text in order to place a particular site higher in the list of returned results. The technique proves extremely effective when blogs are used to roll several links repetitively since updated pages maintain a higher ranking [18].

2.1.2 Semantic Web

Because of the shortcomings of human based and strictly automatic indexing most current research is focusing on agent and meta-data based indexing. This led W3C to start research in the area that has been deemed the Semantic Web. The goal of the semantic web is to add meaning to the structure of the web. This structure could not be found in html since it is a presentation markup language. Its simplicity did not give it the necessary structure to facilitate searches and other complex tasks. To assist service development W3C made the Extensible Markup Language (XML) Recommendation [2]. This markup language allows for the creation of domain or task specific languages. Through its use the Resource Description Framework (RDF), and

semantic web languages like DAML and OIL have been created [17].

2.1.2.1 RDF

The basis of RDF is an object. This object can have attribute and that attribute can have a value. This statement, A(O,V), can be nested or chained. On the web this is important because it provides semantic groups because each object is independent. The nature of RDF also makes expressing a domain model particularly easy. Although RDF is the beginnings of semantic encapsulation on the web, the language does not define primitives that allow for sub-classing and capturing the relationships between objects. To capture these relationships, Ontology Interchange Language (OIL) must be used [16].

2.1.2.2 OIL and DAML

OIL is a web language defined using XML and RDF. OIL is designed to meet several goals including:

- Maximum compatibility with XML and RDF
- Provide primitives for use by large user communities
- Provide formal semantics for machine interpretation

Target fields for its use include search engines, e-commerce, and knowledge management [17]. OIL is an ontology language. This means that it provides a means to pass the formal concept of a domain, its ontology. The use of an ontology allows for machine based communication between two agents [16].

The DARPA Agent Markup Language (DAML) is an ontology language created in a joint project between the US government and the W3C. This language borrows many structures from OIL, but differs in several ways. The major differences between the two are that OIL is far more backwards compatible with RDF Schema, and OILs

logic was designed to allow for complete and efficient reasoning capabilities. Currently the DAML language is being developed in two phases, the ontology language and the logic language [17].

2.2 Human Computer Interaction Research

In the field of Human Computer Interaction several cognitive models have been developed to understand human thinking. The three most common cognitive models are distributed cognition, activity theory, and situated action. In situated action, the unit of analysis is found through the relationship of a person and the environment or arena in which they are acting. This shows that a familiar situation or event usually brings about a common set of actions. In activity theory, the unit of analysis is the activity that is taking place. The subject works toward completing a goal through the use of some object. Context is found through the interaction with people and objects in the environment. Distributed cognition states that the unit of analysis is the entire system rather than the individuals that are part of the system [22].

Distributed cognition gives great weight to the importance of the event taking place since it states the entire system is important as opposed to the individual when it comes to human understanding and actions. Activity theory focuses on the fact that context is based on the relationship between the user and the environment. If some of the context of the event is known, the event context is worth more than the individual context of those who are acting within that event. Situated action supports events as an important part of human thought through its description of the person acting in a certain way in reference to a given environment.

The cognitive models listed above deal only with the brain's reaction to outside stimulus. The stimulus for the brain's information processing comes from three sources: external events, internal events or goals and emotions, and time-driven events

which are external events triggered by external clocks. The internal goals can be considered derived information that is gathered from external stimulus [27].

Another important concept to look at in HCI is the necessity of sensors and memory. After the senses bring in data about an external event, that data is processed through the recognize act process. The data is stored in working memory. Long term memory is accessed in order to understand the data in the working memory. The act takes place when the short term memory is modified by the long term memory in order to facilitate the next action [13]. This research gives weight to the argument that any system designed for information management will need some internal means of data storage and data gathering.

2.3 Experiential Computing and Multi-modal Data

Before media can be semantically linked, it must be collected. An area that deals with multi-modal data collection is experiential computing. Experiential computing encompasses research in several other major research areas including human computer interaction, database development, visualization, and signal processing. This new field of research seeks to bring knowledge to users by creating a environment where they can gain insight through the use of all available data sources. Creating such an environment requires achieving the following data management goals:

- Effective collection of multi-modal data
- Analysis of multi-modal media in a timely manner
- Global Privacy and security settings for a heterogeneous network

Table 1 compares several projects that focus on dealing with some or all of these goals. Although many of the projects in these areas do focus on multi-modal data collection, they do not have the ability to fully integrate and analyze the data their systems collect. This imbalance takes place because a majority of research efforts in

Table 1: Experiential Systems Review

	System Research Issues	Sensor Modalities And Processing	Context Through	Media Integration	Data Modeling, Querying, CQ
Aura [25]	Networking, File Access,Resource Monitoring, Runtime Support	RF-detectors, Video Gesture Recognition, Eye-Tracking,	User Location, Printer Location, Available Bandwidth	No or Non-explicit	No or Non-explicit
Endeavour [1]	Scalable services on ubiquitous infrastructure	MEMS sensor/actuator, cameras, wall-mount displays, PDA, Laptop	User interactions, Data	Non-explicit	Data-streams
Infosphere [19]	Application directed filtering	None/ software detectors for virtual world	User input	No or Non-explicit	No or Non-explicit
Oxygen [5]	Ubiquitous computing infrastructure, network of devices, applications	Audio, Video	Mobile Tags, Location Server, Indoor Tracking	No or Non-explicit	No or Non-explicit
Experiential Meeting Room [24]	Sensor network stack, Large-scale distributed program deployment, debugging, monitoring. Resource adaptation	MULTIPLE: Audio, Video,Mixed-Text	Relations in the data, User Interactions, User/Hardware Location	Yes	Yes

this area focus on the networking and middleware problems and not on data modeling or its impact on information management, access, and assimilation. The last project in the table the Event Meeting system does provide both media integration and querying. This is a parallel work with the project presented in this paper which uses a specific meeting related model to unite media. The proposed project will use a general event-based model and will focus on the implications of that model on the design of a web service architecture and the analytical power of systems designed on the given model.

CHAPTER 3

THE PRESENTED RESEARCH

The initial goal of the WWW was to present information. This can be seen in the development of html. As the web has grown, information management and search issues became far more central issues. The development of XML and the web ontology languages are prime examples. Although the ontology languages will provide machine readable information, they are far too complex for the average user to manage in a web environment. In order to facilitate experiential computing, a simple language needs to be created that focused on the fundamental element of experience: an event. The major contributions this language will provide are extensibility, backwards compatibility, and the combination of domain and attribute based searched. The event web created in this project will be beneficial to web development in several ways. Through its use, information normally found in databases can be stored in document form on the web. This format will give access to information that could not be searched previously because it was located only in databases. The project will also benefit groups like educational institutions because it will provide a modifiable format in which to pass around useful information.

3.1 Event Definition

> *Event: The fundamental entity of observed physical reality represented by a point designated by three coordinates of place and one of time in the space-time continuum postulated by the theory of relativity [4].*

Although this definition of an event is derived from the physics domain, it gives insight into the base design for the system. An event is an occurrence of significance that is observed at a particular time and place through the given information present

[24] [9]. This definition differs from the notion prevalent in distributed systems where events are used to define state transitions or points in time, when the data deviates from a model [15]. An event using the above definition can be any instance that exists in time and space that has a context including a meeting, a speech or any other object that takes place in time and space.

The fundamental challenge that is encountered in the interpretation of such a model stems from a dichotomy in requirements. On one hand the event data model needs to be flexible to adequately represent the many variations in web data content, different contextual aspects associated with specific information, and issues associated with different media syntaxes. On the other, it needs to bridge the heterogeneity associated with information presentation, and emphasize the fundamental aspects that are shared between various types of information. Additionally, two other related issues also need to be kept in consideration. First, there exists a tradeoff between the volume of information represented in a data model and scalability of the systems built around it. Second, the data model should be expressive enough to represent the various relationships that are expressed in the data.

It is important to note that events must have a semi-structured nature in order to match the evolving information space they are modeling. However, several common attributes that are present for all events are defined. These attributes, even if null valued, help to provide context about an event and meet the general definition for an event. The definitions for these attributes are as follows:

Name This is a human comprehensible event identifier. It provides a quick method for humans to differentiate between events.

Time The time for an event is the range in the temporal dimension in which it inhabits. This is a mandatory attribute because it is necessary for the definition of an event.

Location The location for an event is the physical or cyber position of the event. This constraint is larger than a geographic area because many events now take place in cyber realities. Examples include bank transactions and the processing of information on a distributed system like Seti.

Category The category of an event is the classification of the event. This classification is derived from information associated with the event and the prior knowledge involved in defining the event. Optional attributes of events will vary between different categories, but are constrained to be same within a category. Further specifications of categories can create sub-categories whose events inherit the optional attributes of the parent category and define other optional attributes.

Data Includes documents, media and associated information that is relevant to a particular event. It is important for an event because events can be recognized from several different sources. Data provides a link to those sources. It is a multi-valued attribute. It may be noted that values for other mandatory attributes like participants, category, and name can be gathered from data.

Participants The definition for participants comes from the cognitive sciences. Participants are defined as the objects or actors that are interacting with the environment where an event is taking place [22]. Actors are the people taking part in some task. Objects are the elements being acted upon or interacting with the actors and environment. Participants are a requirement for an event because the context for the event may be derived from the participants. Also, this allows a direct way of relating events to people and vice-versa. For example, events could be related by finding if there are common participants in them. Also, information about participants can provide contextual data on events. This is also a multi-valued attribute and is defined to have none, one or more values

because sometimes it may not be possible to determine the participants of an event.

After defining the mandatory attributes of an event, it is possible to define relationships and create structures for events. The current relationships that are defined between events are the parent-child relationship, category relationship and the domain. This is not an exhaustive set of relationships since events are semi-structured.

Parent-Child Relationship This relationship takes place when one event encapsulates another event. An example is a baseball game. At this event many other small events are taking place like vendors selling food. This general relationship becomes apparent through the time and location associated with events. An event that takes place in a large geographic area during a certain time range may be composed of the smaller events that take place in the subsets of that geographic area and time range.

Category Relationship This relationship takes place when events belong to the same category. Events in a category, share the same optional attributes. These can be used as criteria for comparing events within this group. Sub-categories also come into play here because the events in these categories can be compared by the attributes they inherited from their parent categories.

Domain Relationship The domain is a structure in which related categories of events are stored. The relationship between these events is determined by the environment or organization on which the domain is being modelled. An example domain would be a university. This university domain would contain sub-domains for each of its organizational units. The categories within this domain and its sub-domains would contain events related to a university structure like seminars, classes, and meetings. Similar organizations would share category definitions in order to easily compare items between domains.

After defining the mandatory attributes and the event relationships several advantages become apparent. The mandatory attributes allow easy referencing from a semi-structured base to a structured one. This would be important when mapping between graph-based systems and relational systems. The defined event relationships can also be used to perform powerful time, location, and category based queries on events. Another advantage of the event model is that it is flexible enough to be implemented in a variety of ways including those that involve ontology languages or data modelling languages. These different implementation approaches can be used to highlight different strengths of the model.

3.2 Event Detection

Event detection is the ability to query media for information related to a particular type of occurrence. It requires the ability to define an event and translate that definition to a format that various crawlers, signal processing tools, and other media query agents can understand. Detectors can work in conjunction with one another querying several different data types in order to determine if a particular event is taking place. This will allow the system to collect multi-modal data and analyze that data using tools specifically designed for that data type to find particular information. Such detection is necessary for many applications including situational monitoring and environment exploration. At present four types of event detectors have been defined. They are as follows:

Specific Event Detectors The detectors in this group are dedicated to finding a specific user request to find an event meeting certain criteria. They query any possible media stream to fulfill these requests.

Domain related event detectors The detectors in this group are dedicated to finding instances of the events defined in the domain in which they run. These detectors run continually without being queried by users.

Software detectors These detectors are web crawlers. They index web pages and other documents in order to find events.

Hardware detectors These detectors buffer streams from physical sensors and directly query those streams for events.

It is important to note that these groups are not mutually exclusive, but that each category brings its own complexities. The specific event detectors require access control for interaction with users. The domain detectors present a challenge in creating a modular agent to control their actions. The software detectors present a problem in memory management for the indexes they create. The hardware detectors present a problem in sensor management and buffering. The common goal of all these detectors however, is to abstract away the sensors or source from which the media is found.

The event detectors will play an important role when multi-modal data is being collected. Event detectors are designed to query a certain type of media for specific information. Detectors can work in conjunction with one another querying several different data types in order to determine if a particular event is taking place. This will allow the system to collect multi-modal data and analyze that data using tools specifically designed for that data type to find particular information.

A novel use for event detectors is on heterogeneous networks where the detector will determine whether a particular client event storage system has rights to access it. Updates will only be sent to client service networks that have the appropriate security registrations.

3.3 System Design

Along with the event model defined in Section 3.1, the following criteria are considered for the prototype system:

Extensibility Because of the dynamic nature of information and the many possible

ways to detect events, the architecture needs to facilitate run-time assimilation of new components. The communication protocols between system components also need to be extensible to support changes in system modules.

Distributed Information Management With the heterogeneous nature of data, event descriptions can involve distinct types of media including images, video, or audio. This media needs to be stored and processed in an appropriate manner, thus requiring distributed information management.

Selective Persistence Certain domains may require media associated with events to be available all the time. Update agents must be able to effectively cache the media for these domains, without the necessity to make all information persistent.

Search Efficiency Efficient search and processing of event-based data is essential. This is especially challenging because the event's semi-structured nature complicates search performance.

Update Efficiency and Consistency The distributed nature of event-based information management requires updates to be supported efficiently and consistently as simultaneous queries and updates may occur. System updates must not become visible until they have been propagated throughout the entire system, to ensure consistency.

The system architecture created using the event model and these goals is shown in Figure 1. The information flow in the architecture is as follows: The event detectors are used to inspect media that could possibly be of use to the system. When an event is detected, the media is stored, and an event summary is created. This is sent to the update server. The update server uses domain knowledge to update event categories and the eventbase where events are stored.

Figure 2 shows a breakdown of the eventbase. The subdivisions within the eventbase exist in order to enforce a level of privacy and security by separating the data attribute from the rest of the event's attributes. By doing this a system is created where a user must send a request and have that request and user identification validated by the data request server before they can access media associated with the event. This allows a certain base level of information to pass unregulated from the system while maintaining security on all the media associated with that information.

A second perspective of the system in Figure 1 can be described from the client. The client is any program or device that can access the domain or the event data for information. Typically the client accesses the domain to find out about the event categories that are present. The client picks the category they wish to view, and accesses the event category to retrieve the associated events. If particular events are of interest, then a request is sent to the eventbase to get the data pointers for that event. The client then connects to the data store to access the media for the events. If the particular event is not available in the system but may be available at a future point in time, the client can initiate a continuous query. The continuous query server takes the specified time, location, category, and participants from the client request and translates any requests into the form necessary for any given event detector to understand. The event detectors query their individual media sources until they find an event matching the request sent to them or the job is terminated. Once an event is found using the event detectors, the continuous query server returns the information to the client.

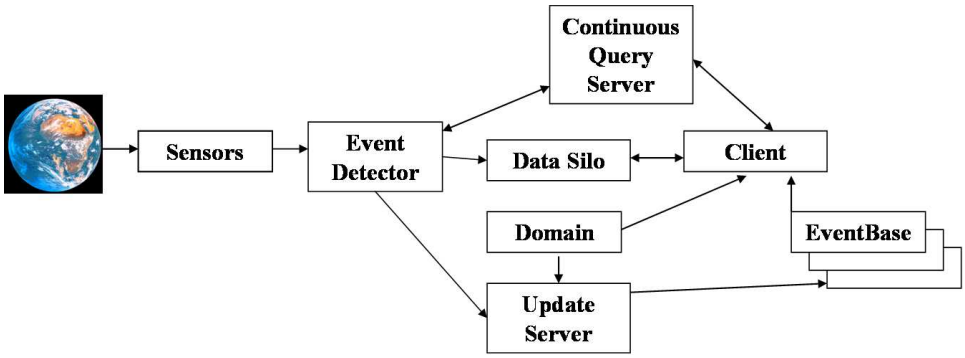

Figure 1: System Description

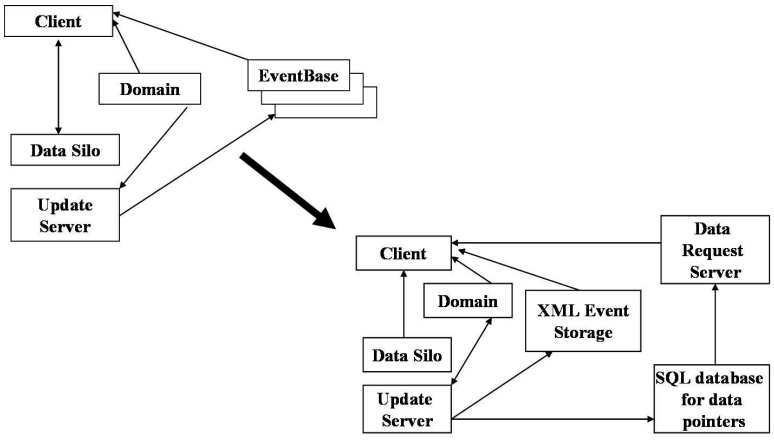

Figure 2: Expanded Eventbase

CHAPTER 4

IMPLEMENTATION

The discussion of the implemented system will start with the implications the event model has on the web architectures, followed by the evolution of the system to its current implementation.

4.1 Implications of the Event Model

Implementation of the event model poses several challenges on system design. Several of the factors to be considered are discussed in the following section:

Extensibility: This challenge comes from the compromises that need to be made in order to store event information. Since events can be considered semi-structured, any method used to store events will need to be able to store additional options without needing to update the system schema. Fortunately contemporary web service development has considerable experience with extensible languages like XML [10], [23],[26]. The downside is that unstructured or semi-structured data has a far lower performance in retrieval compared to data stored in a relational database.

The nodal nature of semi-structured data also poses a security problem in implementation. Contemporary implementations of relational databases allow for quite complex user authorization. Most contemporary extensible database solutions do not contain nodal user authorization. This will make it harder to design systems where certain optional event information is hidden depending on the user authorization level.

For the current implementation, it was decided to take advantage of a combination of both paradigms. Relational databases are used to store information about media

and pointers to event information. Extensible databases are used to store mandatory and optional event attributes. Future work is needed to develop algorithms to determine if repetitive event information should be moved to the relational or XML database.

Transportation and Messaging: The challenge of transportation and messaging of event information and media in the system is most relevant to the event detectors. Event detectors will need to process sensor data both on and offline. At times offline media will not have a correct time stamp and human intervention will be necessary. In the case of online processing, a complex system design with multiple sensors and signal processing units on various processors will require time synchronization. Because of processor drift a system with absolute time synchronization is not possible, but solutions like Media Broker [21] allow for the creation of multiple threads on various machines that will provide transport and time stamping for sensor data. Solutions like the above will also provide a method to add modules that will translate sensor data of one form to another while transportation is taking place. Such a capability will be needed in event based systems where the sensor data produced needs to be down sampled for clients to use effectively.

The current architecture and its use cases are simple enough so that the use of Media Broker and other such techniques are not yet necessary. However, system development and complex use cases where dozens of sensors and processing devices will require its use.

Autonomy: Determining the amount of autonomy modules in the system will have will require measuring several factors. Although event detectors will use signal processing to detect events, they will also determine events by prior events. In some cases an event detector will perform the same signal processing technique to determine several different events. Such a design will probably best be supported by a network of objects consisting of dapplets found in the Infospheres project [14]. Using

dapplets event detector will exist as its own object that can receive, process, and transmit information. Other objects with appropriate security levels will be able to communicate with the given event detector to send it new input, request a result of current processing, or make other state changes.

The addition of the following technique will be added to future releases where various online semi-dependent processing techniques will be in use. Although this addition will add more complexity to the system, it will provide security features that will allow an event detector to communicate with multiple event systems concurrently and delegate information to all systems based upon their security clearance.

4.2 Current Implementation

The current system implementation is based upon Figures 1 and 2. Components of the system are implemented with servlets, XML, and SQL.

4.2.1 Event Implementation

To provide an extensible format to work in, the event model and its relationships are implemented using XML and the XML Schema Language. Two separate schemas have been created for events. The purpose of the first schema is to define the structure of events. The second schema is used to define the structure of domains.

The general event schema is shown in Figure 3. An example XML schema document and an XML document based on that schema are shown in the Appendix in sections B.1 and B.2. The event schema can be extended through the use of other schemas to add additional specification for specific categories of events. By doing this an event inherits all the attributes of its parent events by reference to the parent event's schema. Extra event information can also be added to any given event because the given event schema makes using of the XML "any" tag.

The domain schema is shown in Figure 4. Example domain schema and XML

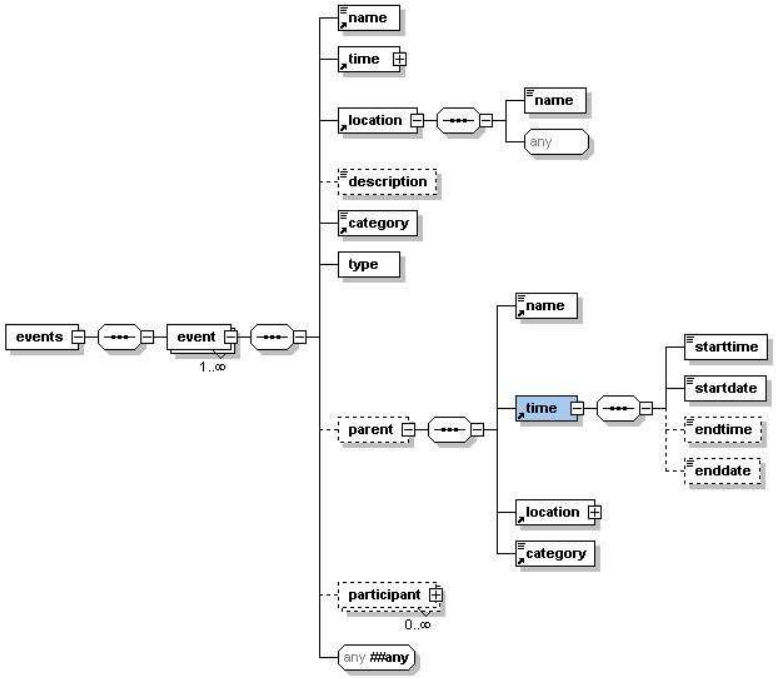

Figure 3: Event Schema

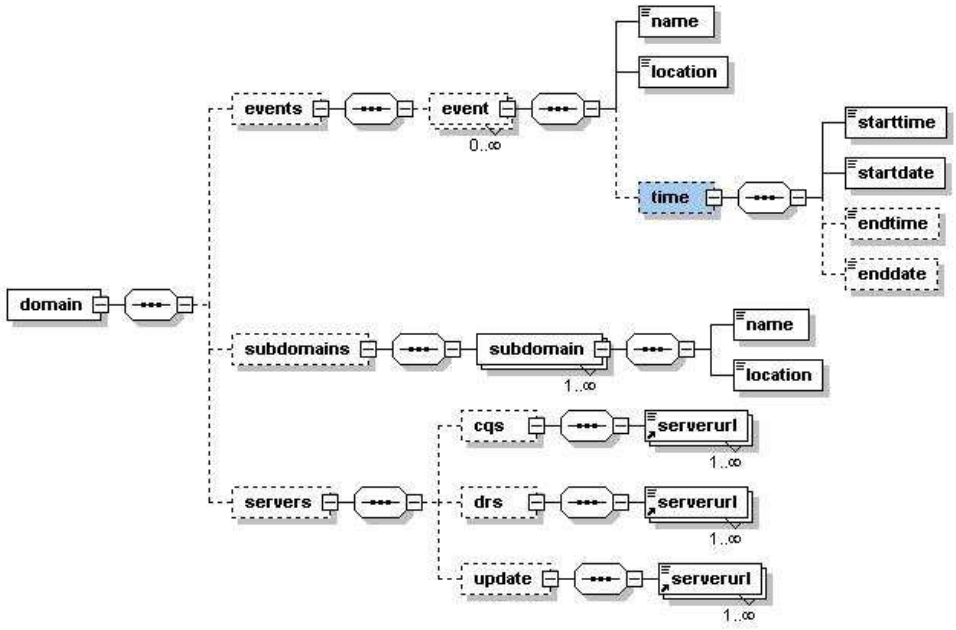

Figure 4: Domain Schema

documents are shown in the Appendix in sections B.3 and B.4 respectively. The extensibility of the domain schema differs from that of the event schema. The domain schema defines the domain element which must be included in any domain.xml document. This element can only be defined once, but it does not necessarily need to be the root tag of the document. This allows the extended domain schema to define elements for use in domain specific information that is not related to the event web. Extension of this schema document is also done through the include and import tags defined by the W3C. No XML attributes are added.

The domain keeps track of more than the categories of events that exist. For this system, the domain tracks the category of the event, the servers that are used for processing events, and sub-domains that are connected to the given domain. Depending on the organization extra elements can be added to the domain document for processing in a particular organization.

4.2.2 SQL and XML Database Implementation

Several storage methods are considered for the XML formatted events. Figure 2 shows the eventbase broken up into several smaller components: event and domain XML and an SQL database for database. This division takes place to ensure data control. A reference to an event is kept in the SQL database along with data pointers that mapped to related events. The schema for the database is found in the Appendix in Section B.5.

Two implementations are considered for storing and querying events. The first method wrote new event updates to XML files on a filesystem. This method had several drawbacks. It was tedious to backup and restore information, and a query method had to be implemented on top of an XML parser to return information about events. After attempting this method, a method using a database was considered. Since the extensibility of an event makes it hard to use in an relational database, an

XML database is used. The eXist database [6] is chosen to store XML data. It is a native XML database that takes advantage of the xml:DB and XUpdate APIs [8] to store and update XML. The database also came with a development implementation of the XQuery specification [7]. Additional features included several APIs to access information in the database. Through the use of this database, it was possible to create an event API on top of XQuery. The database also could be extended through the use of procedures. This made it easier to add features to a domain and queries for new events.

4.2.3 Specialized Servers

The three specialized servers are the data request server, update server, and continuous query server. Each is implemented using the Java Servlet API. The following sections give greater details in the implementation and functions of each.

4.2.3.1 Data Request Server

The data request server makes use of an http connection. This server will not require any session management. The client makes a connection using http and sends the data request through a post. The server reads this data request and responds with the correct information from the database.

Through the use of event XML documents, direct database access are restricted to administrator accesses. The data access server mediates any request where users want to access the other forms of data related to a particular event. XML may be able to show many attributes of an event, but these attributes are text based. When video, audio, or some other media is needed, the database is accessed to gather the related meta-data about other media streams. The event browser does not make a direct connection to the database. It passes a copy of the event that has been selected to the secondary server. It can also specify the types of media that is requested. If no data type is specified, then all pointers to all media are returned. This server contains

```
<events>
    <data_type> </data_type> //0|many
    <event> //1
        <name></name>
        <time> </time> //as defined in event schema
        <location> </location> //as defined in event schema <category>
        </category> //as defined in event schema
    </event>
</events>
```

Figure 5: Data Request Format

a cache of most accessed events. If the information about the requested event is not in the cache, the database is accessed for the event. The server returns the information from the database to the client program where the user can decide which piece of data they wish to view. This method provides added security for the database by letting administrators specify events or event types whose extra data cannot be accessed.

Requests to the data access server are limited to data location requests. The client connection will phrases the request in an XML form (Figure 5. If it is not stated differently in the request format, a tag is used only once in its parent element.

The reply from the server is also in an XML form (Figure 6). It is important to note that the reply can return an empty data element. The root tags could be returned to the client program because the data type that was searched for was not found or access to that particular data is not allowed outside the domain. If it is not stated differently in the reply format, a tag is used only once within its parent element.

4.2.3.2 Continuous Query Server

The continuous query server takes care of long standing requests. These requests are generated when a user searches for an event and is not able to find the results they wish. The user is then given an option to start a continuous query. Continuous queries can also be started if only partial information about an event is known. The

```
<data>
    <element>// [0|many instances]
        <data_type> </data_type>
        <location></location>
        <filename> </filename>
    </element>
</data>
```

Figure 6: Data Server Response Format

```
<events>
    <event> //1
        <name></name>  //possible name
        <time> </time> //as defined in schema section
        <location> </location> //as defined in schema section
        <category> </category> //as defined in schema section
        <participant> </participant> [0|many]
    </event>
</events>
```

Figure 7: Query Server Request/Response Format

client/server setup connection is defined as follows:

- When a new request is sent to the server, a cookie is sent to the client as an identifier. The timeout for the cookie is determined by the server.
- Jobs sent to the server will continue until the jobs either timeout on the server or are killed by the client program. When timeout occurs, the data for the search is saved till the user accesses it or twice the timeout period has taken place. The information for killed jobs is automatically deleted.

Client requests are similar to the request in the data request server. The main difference is that this format does not allow you to specify data types in the search. The response format for this server is the same as the request format. The format is shown in Figure 7

It is important to note that the continuous query server presented here is different from the concept in the database community. The continuous query server

in databases involves retrieving features from continuous media streams and recording those features to alphanumeric streams that can be stored in a database and queried. In this architecture, the proposed continuous query server will interact with such lower level feature detectors in order to service its request. Implementation constraints made it hard to test this particular server on continuous feature detectors. For testing of the server's command set, the server was tied to an event detector attached to the SQL database. After other hard coded event detectors found an event, the information was sent back to the continuous query server which notified the user. The following guidelines are used in the implementation of the server. Time ranges are considered intervals. Category and participant values are considered exact. Name values can be left undefined.

Commands and requests are sent to the server using the post method. The post begins with the attribute of command. The commands that can be sent to the server are listed in Table 2 on page 28. The formats for how information is returned to the server are shown in Appendix A.

4.2.3.3 Update Server

Because of the heavy number of updates that take place in the system, an update server is required. This server is used to commit batch updates and additions to the XML and SQL database. Authentication initially take place outside this server. As with the other servers, the post method is used to move data to the server. The attribute for the post is "command". Three responses can be expected back from this server. The three possibilities are as follows:

1. <uncaughtexception></uncaughtexception>–Some type of failure in processing has taken place. Your event has not been updated. Error message will appear in tags.

2. <postnotsent/>–The sender did not send a post.

3. <postcomplete/>–The post has been accepted and is being processed. Server will make update.

The server continues processing after sending a postcomplete signal. The following actions must take place:

1. The server checks the format of the request then attempts to update the database. If the format is incorrect the update is deemed invalid. The update to the database is accomplished by parsing the XML request into an SQL query.

2. A successful update to the database is followed by an update to the event nodes. The update to the event node is accomplished by looking up the domain the Update Server is on and checking the category information of the update against the category nodes of the domain. If a category match is found, then the update is saved to that node. If a category match is not found, then the update is saved to the event category node. Upon successful notification that the updates to the database and the event nodes have taken place, the update server flags the event so that it becomes readable by the rest of the system.

3. After the event node update, any domain specific processing takes place.

There are several things to note about the update request format (Figure 8. The "original" element must be filled. If it is not filled the request cannot be serviced. At the very least, this means there must be a valid "name", "time", "location", and "category" elements. In the cases where the "participant" or "data" tags are added to the "original" entity, these values will be removed and replaced by the corresponding value or values in the "update" element. The update element does not need to have all elements filled. The only elements that are necessary are those that will change. If participants or data are only to be added (instead of some being removed) then there will only be tags for these items in the update element.

```
<event type="update">
    <original>
        <name></name>
        <time></time>--as defined in event schema
        <location></location>--as defined in event schema
        <category></category>--as defined in event schema
        <participant></participant> --[0|many]
        <data>
            <data_type></data_type>
            <url></url>
            <filename></filename>
        </data>--[0/many]
    </original>
    <update>
        <name></name>--[0|1]
        <time></time>--as defined in event schema [0|1]
        <location></location> --as defined in event schema [0|1]
        <category></category> --as defined in event schema [0|1]
        <data>
            <data_type></data_type>
            <url></url>
            <filename></filename>
            <time></time>
           <location></location>
        </data>--[0|many]
        <participant>
            <title>
            <type>
            <time></time>
            <location></location>
            <data>
                <data_type> </data_type>
                <url></url>
                <filename> </filename>
            </data>--[0|many]
        </participant>--[0|many]
    </update>
</event>
```

Figure 8: Event Update Request

Table 2: Continuous Query Server Commands

Client Command	Description	Server Response
get timeout	Requests the timeout for jobs on the server	Returns the amount of time a server will process a job before killing it. Timeout is sent in hh:mm:ss format
get server info	Requests server information. The required server information will be specified later in the document	Server returns server information.
Return info	Requests all jobs or a particular job that have returned information	Returns data in request format if there is data or returns the empty request format
Kill	Kills the present job.	Any stored meta-data is sent to the client. Search thread is killed
request	Client sends a request in request format to the server. A CID (cookie) is sent to the client connection.	Server ignores a request if the client sends a CID. If no CID is set, the server accepts the request. Empty response format is returned to the client

Another difference between the participant tags in the update element is the fact that they may also encapsulate a time, location, and data element. The data tag is used to reference data with a particular participant. The location tag is used to relate the participant to a particular location during the given event. Data tags can also have a time and location elements if they are not enclosed by a participants tag. These extra tags in the update element are used to form other relations in the database.

Requests to add new events will match the event schema format in Appendix **??** with the event element containing an attribute called "type" with the value "addition".

The update server connected to the SQL database using the java.sql API. It connects to the XML database through the use of the XML:DB API. Eventual improvements will add support for the Reflections API so no recompilation will be necessary

to add other sources where updates will be.

CHAPTER 5

EXPERIMENTS

The use of events with web services produce many possible areas for testings. In this case, qualitative tests are presented using the system to show what possible systems can be created using it. Quantitative experiments are given to show the analytical ability events provide a user.

5.1 Qualitative Experiments

This section presents three experiments. The first presents experiments on a News Event System. The second shows an example of a query on the system. The third presents a Lecture Event System.

5.1.1 News Event System

The news event system was created using event detectors that monitored Google News and created events based on the news articles. The sensor used in this case took a snapshot of the headlines section on Google News site at 2pm each day over the period of a week and updated the system with the information. The sensor made sure all the news items had a basis in time, location, category and a participant in the form of a publishing agent. An event detector was created to monitor for new events and looked for specific keywords, time ranges and locations. This detector was designed to serve two purposes. It can be used to automatically group incoming information into correct categories and to derive subcategories of these categories. It can also be used by users to create personal event groups and to continuously query the event system for specific incoming events.

As proof of concept for the event based system the aggregated news items were

Figure 9: Event News Results

queried to find events on the California wildfire between November 4th and 6th. Some of the information is shown in Figure 9. Table 3 gives several query formulations that are used to retrieve the information found in the event system from Google News. The snapshot system returned three results for the given query. The results came from MSN, News 24 in Houston and the Xinhau news service. The information that was crawled was semi-automatically processed to verified that these were the only correct results. This case can be compared to the example in the introduction. When querying on Google news for the same information, limited results were returned because of a query syntax problem and the inability to directly query for time and location. In order to return all the information in the given time range on Google News the user would have to pose the query on the keywords alone then conduct a time relevant sort. Unfortunately the number of results required the user to look through several pages of information that take place after the required time range before they can find the exact range they are looking for. The system allows for

Query	Google News
California wildfire November 4 5 6	0 relevant articles amongst 2 retrieved results
California wildfire	3 relevant articles on the first page
California wildfire [time sorted]	0 relevant articles on first page. Valid results (over 30 pages of results in the time range of interest) start around 30th page of results
California wildfire November 4	0 relevant articles out of 12 retrieved results

Table 3: Results for Various Searches in Google

querying directly by date, time, and location along with the keyword search ability that Google provides. The event based system also allowed searches by information source. If a user preferred CNN for their news, they could query for information related to the California fires in a given time range that was produced by CNN.

It may be noted that Google News returned the same number of results in some occasions as the event based system. This is a coincidence created by the discrete sampling the event system used in order to find news articles. If the event based system was given the entire set of articles collected by Google in the given time period, the number of returned results would have been higher. Since Google does not provide access to its system in such a manner to retrieve all such results, it was only possible to use a subset of the results.

A second experiment takes place using the News Event System. It involved collecting media sources from the internet about Iraq for four weeks. Although there were several ways to structure the information, it is decided to create a domain about news, and a sub-domain about Iraq (Figure 10). This sub-domain contained categories of information related to Iraq. The decision for domains was made to ensure that other categories or sub-domains of important news topics could be added without having to change the information being collected on Iraq. The news domain could be searched and matching event categories or sub-domains would be returned to the

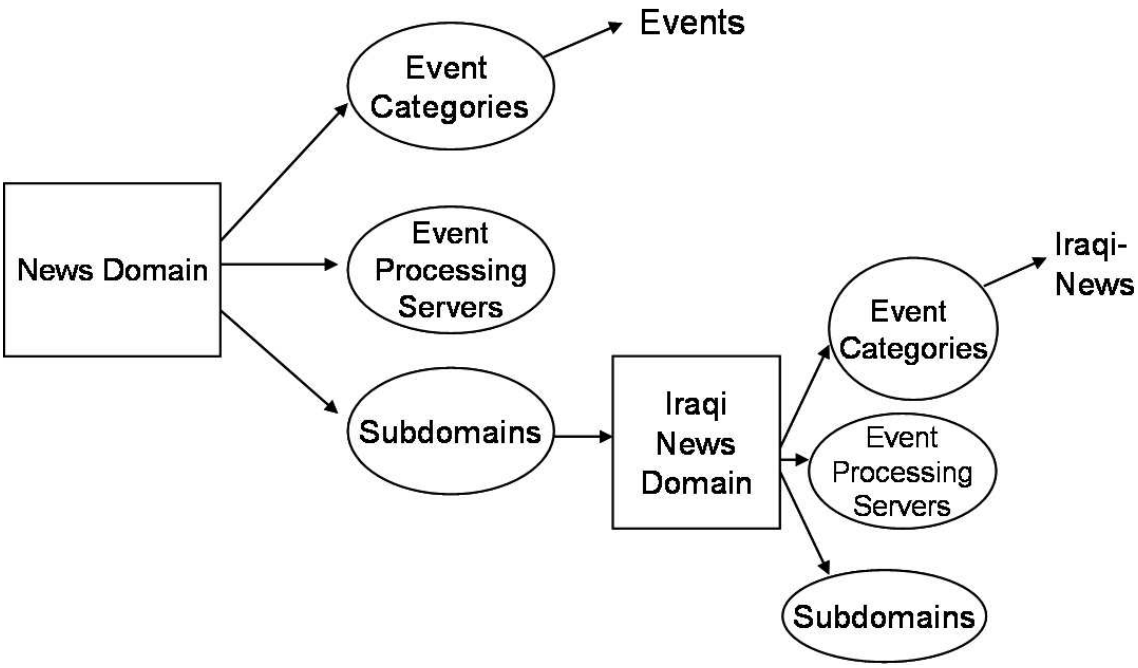

Figure 10: News Domain

user.

The final results from the crawler provided over 2000 pieces of information on world news and around 80 items on Iraq. By placing the information in an event model, the complexity of many types of queries is reduced. One example would be a query about Iraq during a certain time period, e.g.: Return all references to Iraq between November 5th and November 10th. Figure 11 shows the distribution of events when this query is applied to various countries for the given time period. This event based approach to news could be used to show trends in news coverage at given periods of time.

Using Google News the above query cannot be directly posed. One can sort by time relevance but this requires time consuming manual searching of the data for the information one wants. The event model returns all results that take place during the specified time period and requires no manual searching. Besides time based searches, the event model allowed searches on location, web source, and other optional attributes. The additional ability to search by these attributes made it far

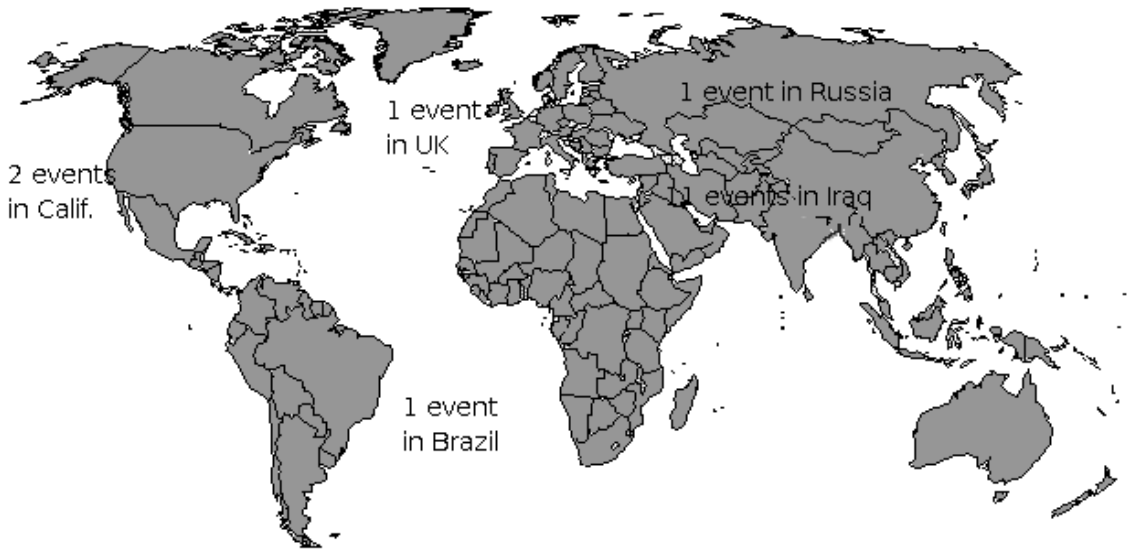

Figure 11: Results of News Query on Various Locations

easier to search on subsets of events that exist within the larger body of events. The system implementation also provides a necessary separation between the meta-data and the media. This gives the user the ability to decide whether the particular topic is important enough to continue viewing before increasing the system load by viewing the media itself.

5.1.2 Continuous Query

The second case study is an example from the domain implementation for Georgia Tech. This case study is chosen because several activities at a research university like seminars, classes and laboratories provide ample information and media streams on which to design an event based system. The hierarchical structure of the university also provides a good basis to model a domain. Figure 12 shows a subset of the domain structure.

It is important to note some of the interesting relationships within this domain.

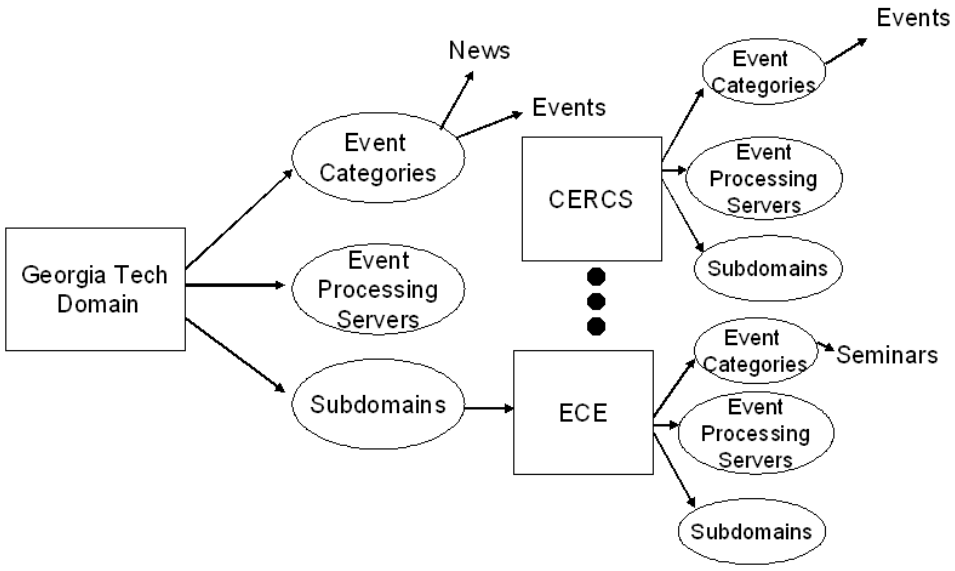

Figure 12: Possible Georgia Tech Domain Structure

The categories in the figure are derived from the same event schema, and the domain and its sub-domains share the same event processing servers. The event categories were updated using an event detector that gathered information from multiple sources. One scenario that was run in testing this system was a continuous query. The user poses the query shown in Figure 13. This query looks for events between 7am and 8pm on November 10 that take place in the Van Leer building and have Lamar West as a participant. The query was posed looking for Lamar West as a participant because he occasionally speaks at graduate level seminars in the Electrical and Computer Engineering School. This is a real-world query that might be run by a graduate student who has forgotten the exact time of the seminar, but remembers the speaker. Upon receipt of this request, the continuous query server would access the database to determine if the domain contained a seminar category. If so the server would query to see if any events matching the required criterion existed. In this case an event matching the requirements did exist. It was an ECE Student seminar on high speed internet access. If nothing was found matching the event the server would

```
<events>
    <event>
        <time>
            <starttime>07:00:00</starttime>
            <startdate>11/10/2003</startdate>
            <endtime>20:00:00</endtime>
            <enddate>11/10  /2003</enddate>
        </time>
        <location>
            <name>Van Leer</name>
        </location>
        <category>Seminar</category>
        <participants>
            <title>Lamar West</title>
            <type>human</type>
        </participants>
    </event>
</events>
```

Figure 13: Continuous Query

continue to monitor the database for the event and activate the necessary event detectors to query the media sources for information matching this event until the job timed out or was killed by the user.

5.1.3 Lecture Event System

The first deals with event-based modeling of multimedia information from an instructional/educational setting. At the Georgia Institute of Technology the teleconferencing research group produces multimedia presentations of lectures and places a semester of lectures from a particular class on a CD for distribution. Typically the information includes audio, video, and powerpoint slides. This media is placed in a set of SMIL [6] presentations where each presentation corresponds with a particular lecture. Each SMIL presentation is used to advance the powerpoint slides in parallel with the audio and the video. The user is also given the option to move to certain positions within the presentation by clicking on any of the topics of the presentation.

After examining the SMIL presentations (Figure 14 (left panel)) for the lectures several advantages can be found if an event based model is used. Although each SMIL presentation encapsulates a lecture (a conceptual notion that embodies some of the characteristics of an event), it places the media in a fixed interpretation with limited querying ability. The event model provides more flexibility by providing different granularity layers. Searches can be issued for information about particular lectures or topics within a lecture. Another advantage of the event model is its explicit use of time. The SMIL presentations are only related by the order in which they took place. Since the event model uses an exact time, searches could be done using time as an attribute.

This multimedia data is modeled using an event-based approach. Two event categories are defined. The first category is lectures and the second is topics. Topics are defined as sub-events of lectures. Each multimedia presentation coincides with a lecture event. The time, location, and name of the presentation are added to the lecture event, and the topics within the presentation are parsed out from the SMIL presentation for the topic events. This provides a name, time and location for each topic event also. A pointer to the presentation is considered data for the lecture. The data for the topic events are the individual pieces of media that make up the presentation.

Using an event based model, it becomes possible to pose many complex queries that were difficult or impossible to formulate earlier, for example:

- Search for particular media within a time range on a particular topic, e.g.: Search for all slides on the topic of Maxwell's or Wave Equations in the first two weeks of lectures.

- Search for particular media on a given date about a particular topic, e.g.: Search for the first video clip in a presentation that is about Maxwell's equation.

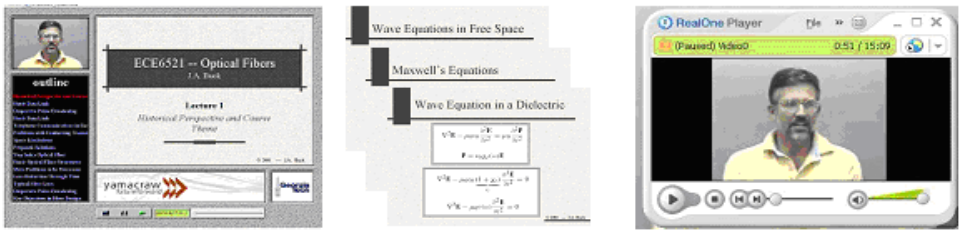

Figure 14: (Left) Original Multimedia data. (Middle) Results from Query 1 (Right) Results from query 2

The first example query above returned 11 slides that referenced wave or Maxwell's equations. A few of the slides are shown in Figure 14 in the middle panel. The results for the second example query are shown in Figure 14 in the right panel. For the latter query, the SMIL file was parsed by times and the video segment referring to the event was isolated.

5.2 Quantitative Experiments

To provide an adequate test for the system described in this paper, the media from a research meeting was used to create events. This media included mpeg files with both audio and video and a PowerPoint presentation. Four event categories were defined for the experiment: meetings, presentations, slide-transitions, and speaker-changes. Figure 15 shows the parent-child relationships that exist between these events in these categories.

It was decided to manually extract events from the given media. However, rules were created for the manual extraction of events to better simulate the extraction of events through an automated manner using various processing techniques. A set of extraction rules was associated with each of the following event categories.

1. Meeting events are determined by checking calendaring systems along with the remote sensor feeds that are transmitted from the location of the meeting. The calendaring system provides time, location, and a possible name for the event. Since the sensors in the meeting room are only operating when the meeting

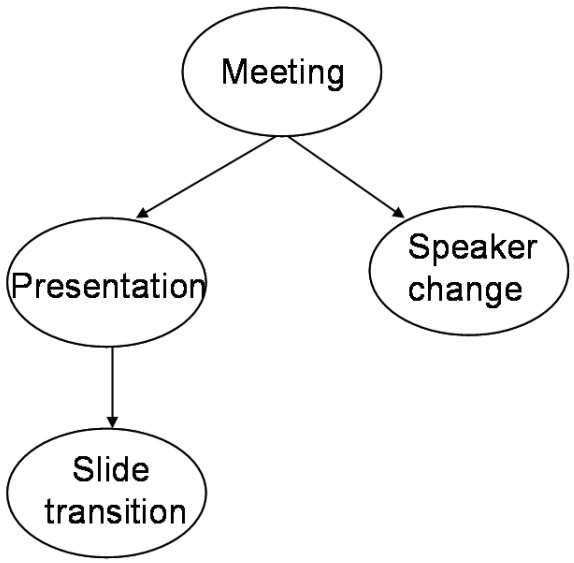

Figure 15: Event Relationships

is taking place, the sensors provide a time span for the meeting along with media that can be processed to find other types of events and participants in the meeting.

2. Speaker-changing events are determined from the mpg feed that comes from the sensors. A new speaker is defined by fifteen seconds of continuous audio from a given speaker. This is used to simulate a speaker recognition engine that has stored audio values for the given participants.

3. Presentation events are determined from checking whether the projector is on and if a PowerPoint presentation has been opened. This can be done through automation by checking the video feed for a certain light threshold and enabling special ActiveX programs on the presentation computer that transmit information when a PowerPoint slide show begins.

4. Slide-transition events are defined by the amount of time spent on a given slide. This can be determined from the video stream by setting a certain threshold for pixel changes in the presentation area. When this threshold changes, a new

Event Category	Media	Number of Events
Meeting	mpg, ppt	1
Presentation	ppt	1
Speaker-change	mpg	35
Slide-transition	png	8

Table 4: Event Statistics

```
for $i in document('/db/GATECH/experiment/newslide.xml')//event
for $k in document('/db/GATECH/experiment/speaker.xml')//event
where $k/time/starttime >= $i/time/starttime and
$k/time/starttime<=$i/time/endtime or
$k/time/endtime>=$i/time/starttime and
$k/time/endtime<=$i/tiime/endtime or
$k/time/starttime<=$i/time/starttime and
$k/time/endtime>=$i/time/endtime order by $i/name return <group>
    {$i/name}
    {$k/name}
    {$k/participant}
</group>
```

Figure 16: Query to determine speakers during slide events

slide must have been displayed. Slide-transition can also be determined by using ActiveX programs on the presentation machine to monitor when a click takes place in the presentation. This method can also be used to further parse the PowerPoint presentation by creating a screen shot of every slide and saving it as an image file so that each slide can be accessed individually.

After annotating the events, and archiving the associated media, it became possible to see interesting trends in the meeting through the use of events. Table 4 contains statistics for the meeting that was parsed into events.

The simplest queries that can be posed by the system include requesting all media associated with an event or selecting an event of a given type during a certain time range. The query illustrated in Figure 16 is a more advanced query that can be used to return results that will help in the analysis of the meeting. This query is posed using the XQuery specification from the W3C.

Figures 17, 18, and 19 show the result of several queries run on the systems. Figure17 is the result of a query that is used to find the maximum amount of time spent on each slide during the meeting. From this figure one sees that slide 6 had the longest associated discussion. Figure18 shows the results of the query found in Figure 16. This graph shows the number of participants who were speaking when a particular slide was shown. Figure 19 shows the results of extending the query in Figure 16 to return the media associated with the event where the most participants are talking.

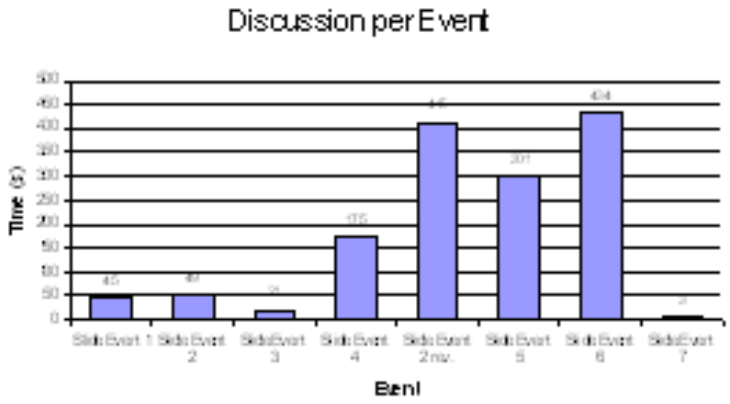

Figure 17: Discussion Length per Event

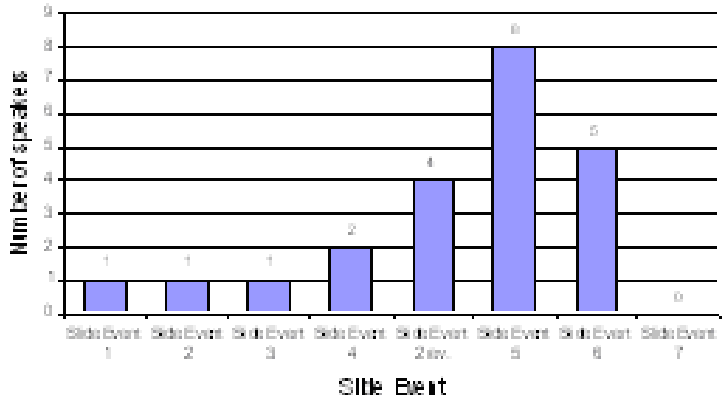

Figure 18: Speakers per Event

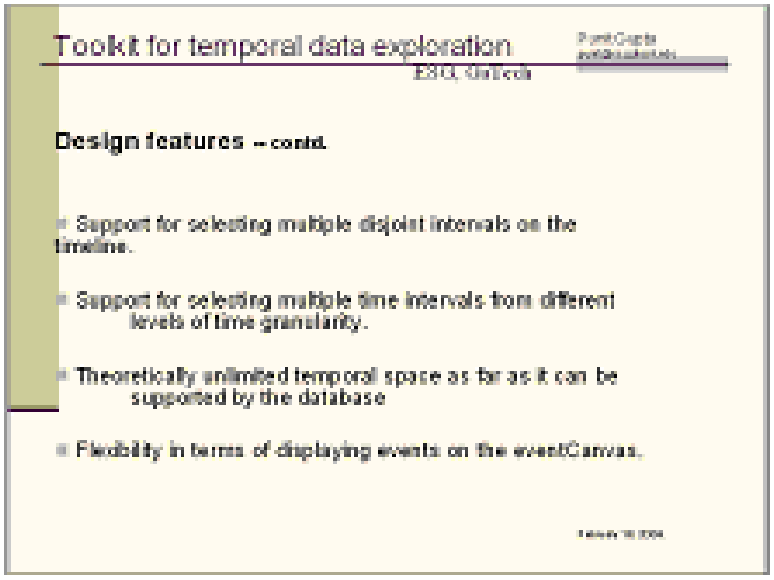

Figure 19: Slide with most active speakers

CHAPTER 6

CONCLUSION

The research presented in this thesis focused on the definition and implementation of an event based system for a web architecture. The system allows for multiple methods of event detection to be used in order to create an event summary. The initial system testing also shows that events are useful in the analysis of information. In order to make events as useful as possible in analyzing information, several areas of further research are necessary. Chief among these areas are event detection, further event model development and dynamic system upgrades.

Event detection can be extended in many areas. Throughout the thesis, several event detectors were discussed. All these detectors were to some extend hard coded with information about the environment in which they worked. Although detectors acted on many different types of media, they could not access the domain information for the event categories they were updated or update multiple domains at the same time by detecting multiple events from the same media source. By eventual necessity, event detectors will need to evolve into intelligent agents. These agents will be tasked to detect events along with other considerations like security. The eventual goal for event detectors is to place them on the web as an interface to sensor(s) where the detector received requests for events from an event based system, downloaded schemas for those events, checked to see whether any sensors could return that given information, and return information to event based systems that had the clearance to access the given sensors. Such a component will need to be modular, dynamically updated, and have a high measure of fault tolerance.

After considering event detection, the event model is the next and a continuing

area of research. The event model presented in this paper is in its initial development. With any such broad model many use cases need to be considered in order to prove the completeness of the model. At present, two areas of major work in the model are the considerations of time and locality for space. The implication of point and interval based events can make for different types of system design. The current implementation focuses on interval based events, but future work will need to be spent on point based events and the effects this will have on the relationships between events. The problem of spatial locality will also need to be addressed. Since the event based system has been approached from the individual application at this point, locality did not play a great factor. When individual applications do not play such a great application or when needing to determine events related by area, spacial locality will become a greater issue. Even in the current implementation latitude and longitude were considered for spatial coordinates, but these systems do not hold when dealing with sensor data not based on the surface of the earth.

One of the last areas of continuing research will be dynamic system upgrades. This requirement is necessary for more than just event detectors. The modularity of the system components will help in this area. One area where dynamic updates will be necessary is the update server. The usefulness of this component will also grow with the ability to register and update multiple eventbases. The development of a registration system that can access an eventbase's knowledge and return results in the necessary format for that eventbase will be useful. In order to meet such a requirement, it may be necessary to move further into the area of Semantic web research and create an ontology of categories. Such techniques weaken when the ontology grows to large. One area of research may be in creating an ontology for a set of related events with given synonyms. When a certain event is updated with a given category, extra information could be sent detailing other closely related events to individualize that event so it will only be updated to the correct areas.

Dynamic updates will also play an important role in the continuous query server. A registrar will need to be created between the server and the event detectors. It will need to be dynamic in order to handle the failure and addition of event detectors.

These are some of the future areas of research for event based systems in web architectures. By addressing these problems and those underlying this initial work, systems will begin to bridge the connection between the web and the real world.

APPENDIX A

CONTINUOUS QUERY SERVER RESPONSES

```
<server>
    <name> </name>                --server host name
    <backup></backup> [0|many]    --alternative server if present one is down.
    <timeout></timeout>[0|1]      --in XML Schema time format.  Process timeout
    <maxjobs></maxjobs>           --maximum number of jobs
</server>
```

Figure 20: Server Information Response

```
<server>
    <timeout> </timeout>         --in XML Schema time format.
    Process timeout
</server>
```

Figure 21: Server Timeout Response

APPENDIX B

EVENT IMPLEMENTATION EXAMPLES

B.1 Event Schema Document

```
<?xml version="1.0" encoding="ISO-8859-1"?> <!-- edited with
XMLSPY v5 rel. 4 U (http://www.xmlspy.com) by Derik L Pack (Ga
Tech Research Group) --> <xs:schema
targetNamespace="http://eventweb.oit.gatech.edu:8084"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://eventweb.oit.gatech.edu:8084"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="events">
     <xs:complexType>
          <xs:sequence>
              <xs:element name="event" maxOccurs="unbounded">
                  <xs:complexType>
                      <xs:sequence>
                          <xs:element ref="name"/>
                          <xs:element ref="time"/>
                          <xs:element ref="location"/>
                          <xs:element name="description"
                            type="xs:string" minOccurs="0"/>
                          <xs:element ref="category"/>
                          <xs:element name="type"/>
                          <xs:element name="parent" minOccurs="0">
```

```
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element ref="name"/>
                                <xs:element ref="time"/>
                                <xs:element ref="location"/>
                                <xs:element ref="category"/>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="participant"
                      minOccurs="0" maxOccurs="unbounded">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element name="title"
                                  type="xs:string"/>
                                <xs:element name="type"
                                  type="xs:string"/>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                    <xs:any namespace="##any"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
</xs:element>
```

```
  <xs:element name="time">
      <xs:complexType>
          <xs:sequence>
              <xs:element name="starttime" type="xs:time"/>
              <xs:element name="startdate" type="xs:date"/>
              <xs:element name="endtime" type="xs:time" minOccurs="0"/>
              <xs:element name="enddate" type="xs:date" minOccurs="0"/>
          </xs:sequence>
      </xs:complexType>
  </xs:element>
  <xs:element name="location">
      <xs:complexType>
          <xs:sequence>
              <xs:element name="name" type="xs:string"/>
              <xs:any/>
          </xs:sequence>
      </xs:complexType>
  </xs:element>
  <xs:element name="name" type="xs:string"/>
  <xs:element name="category" type="xs:string"/>
</xs:schema>
```

B.2 Event XML Document

```
<events xsi="http://www.w3.org/2001/XMLSchema-instance"
schemaLocation="http://eventweb.oit.gatech.edu:8084/xml/eventschema.xsd">
    <event>
        <name>Meeting-Punit-speaker</name>
        <time>
            <starttime>17:15:00</starttime>
            <startdate>2004-02-18</startdate>
            <endtime>18:13:44</endtime>
            <enddate>2004-02-18</enddate>
        </time>
        <location>
            <name>TSRB</name>
        </location>
        <category>meeting</category>
        <participant>
            <title>Derik Pack</title>
            <type>human</type>
        </participant>
        <participant>
            <title>Rahul Singh</title>
            <type>human</type>
        </participant>
        <participant>
            <title>Rachel Knickmeyer</title>
            <type>human</type>
        </participant>
```

```
<participant>
    <title>Punit</title>
    <type>human</type>
</participant>
<participant>
    <title>Zhao Li</title>
    <type>human</type>
</participant>
<participant>
    <title>Bo Gong</title>
    <type>human</type>
</participant>
<participant>
    <title>Pilho Kim</title>
    <type>human</type>
</participant>
<participant>
    <title>Jenny Lee</title>
    <type>human</type>
</participant>
<participant>
    <title>Derek C</title>
    <type>human</type>
</participant>
<participant>
    <title>Cartig</title>
    <type>human</type>
```

```
        </participant>
        <participant>
            <title>Bin Lui</title>
            <type>human</type>
        </participant>
        <data>
            <data_type>ppt</data_type>
            <url>http://eventweb.oit.gatech.edu:8084/data/</url>
            <filename>PowerPointSlice.ppt</filename>
        </data>
    </event>
</events>
```

B.3 Domain Schema Document

```
<?xml version="1.0" encoding="ISO-8859-1"?> <!-- edited with
XMLSPY v5 rel. 4 U (http://www.xmlspy.com) by Derik L Pack (Ga
Tech Research Group) --> <xs:schema
targetNamespace="http://eventweb.oit.gatech.edu:8084"
xmlns="http://eventweb.oit.gatech.edu:8084"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
 <xs:element name="domain">
     <xs:complexType>
         <xs:sequence>
             <xs:element name="events" minOccurs="0">
                 <xs:complexType>
                     <xs:sequence>
                         <xs:element name="event"
                           minOccurs="0" maxOccurs="unbounded">
                             <xs:complexType>
                                 <xs:sequence>
                                     <xs:element name="name"
                                      type="xs:string"/>
                                     <xs:element name="location"
                                      type="xs:anyURI"/>
                                     <xs:element name="time"
                                      minOccurs="0">
                                         <xs:complexType>
                                             <xs:sequence>
                                                 <xs:element name="starttime"
```

```
                                        type="xs:time"/>
                                    <xs:element name="startdate"
                                        type="xs:date"/>
                                    <xs:element name="endtime"
                                        type="xs:time"
                                        minOccurs="0"/>
                                    <xs:element name="enddate"
                                        type="xs:date"
                                        minOccurs="0"/>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="subdomains" minOccurs="0">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="subdomain"
                maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="name"
                            type="xs:string"/>
```

```
                        <xs:element name="location"
                         type="xs:anyURI"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="servers" minOccurs="0">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="cqs" minOccurs="0">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element ref="serverurl"
                         maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="drs" minOccurs="0">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element ref="serverurl"
                         maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
```

```
                        <xs:element name="update" minOccurs="0">
                            <xs:complexType>
                                <xs:sequence>
                                    <xs:element ref="serverurl"
                                     maxOccurs="unbounded"/>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
  </xs:element>
 <xs:element name="serverurl" type="xs:anyURI"/>
</xs:schema>
```

B.4 Domain XML Document

```
<?xml version="1.0" encoding="UTF-8"?> <!-- edited with XMLSPY v5
rel. 4 U (http://www.xmlspy.com) by Derik L Pack (Ga Tech Research
Group) --> <domain schemaLocation="domain.xsd"> <events>
    <event>
        <name>event</name>
        <location>http://eventweb.esg.gatech.edu:8084/
                    xml/event.xml</location>
        <time>
            <starttime>01:02:00</starttime>
            <startdate>2000-01-01</startdate>
            <endtime>01:02:00</endtime>
            <enddate>2000-01-01</enddate>
        </time>
    </event>
    <event>
        <name>News</name>
        <location>http://eventweb.esg.gatech.edu:8084/
                    xml/news.xml</location>
    </event>
</events> <servers>
    <cqs>
        <serverurl>http://eventweb.esg.gatech.edu:8089/
                    ContinuousQuery/QueryServlet</serverurl>
    </cqs>
    <drs>
        <serverurl>http://eventweb.esg.gatech.edu:8089/
```

```
                    DataRequest/DataRequestServlet</serverurl>
    </drs>
    <update>
        <serverurl>http://eventweb.esg.gatech.edu:8089/
                    Update/UpdateServlet</serverurl>
    </update>
</servers> </domain>
```

B.5 Database Schema

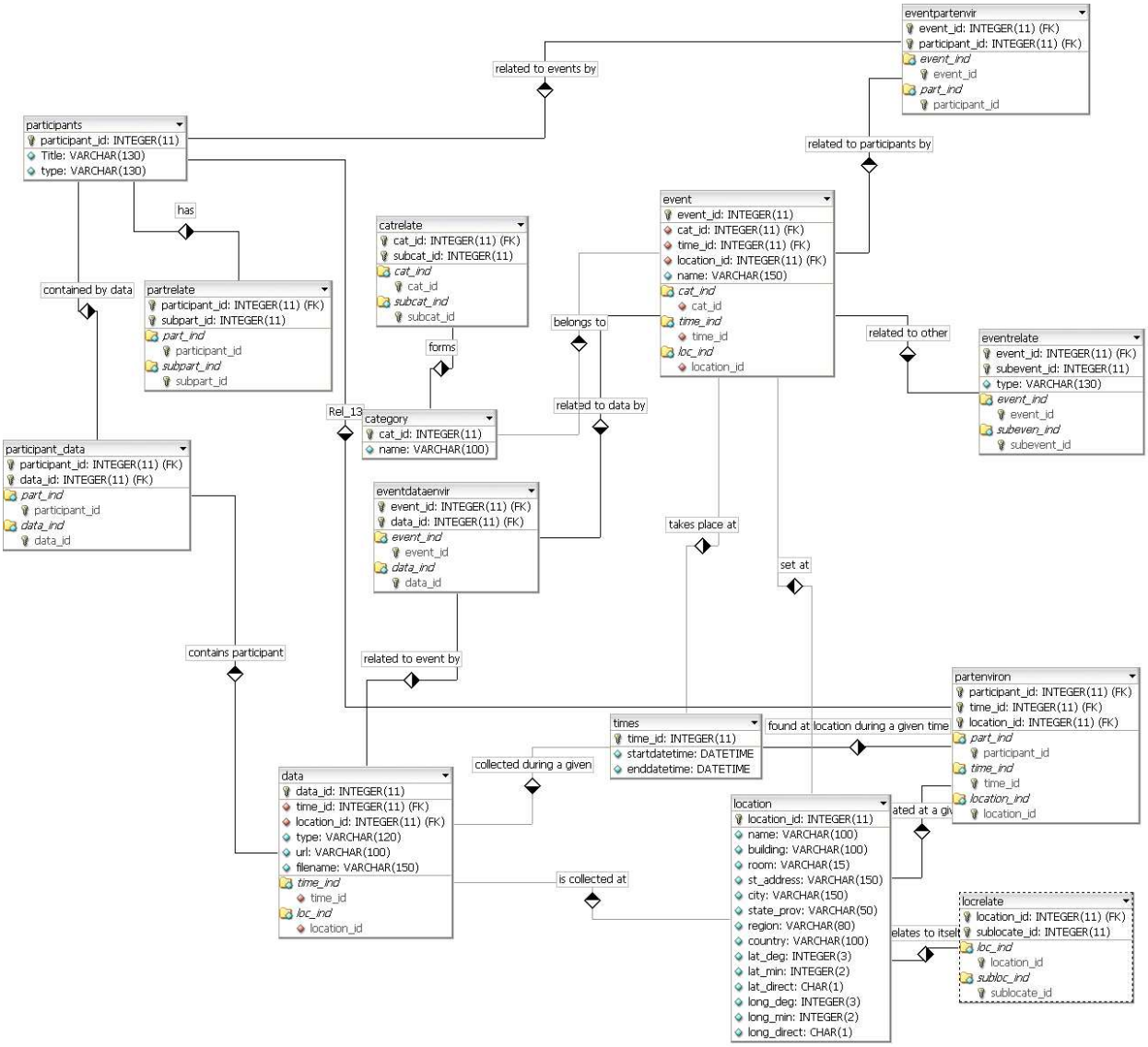

Figure 22: SQL Database Schema

REFERENCES

[1] "The endeavour expedition: Charting the fluid information utility." http://endeavour.cs.berkeley.edu/proposal/.

[2] "Extensible markup language (xml) 1.0 (second edition)." http://www.w3.org/TR/REC-xml.

[3] "A little history of the world wide web." http://www.w3.org/History.html.

[4] "Merriam-webster online." http://www.m-w.com/.

[5] "Mit-project oxygen: Persuasive, human centered computing." http://oxygen.lcs.mit.edu.

[6] "Open source xml database." http://exist.sourceforge.net/.

[7] "Xquery 1.0: An xml query language." http://www.w3.org/TR/xquery/.

[8] "Xupdate use cases." http://www.xmldatabases.org/projects/XUpdate-UseCases/.

[9] Boll, S. and Westermann, U., "Mediaether- an event space for context-aware multimedia experiences," in *To Appear*, 2004.

[10] Bosworth, A., "Developing web services," in *Data Engineering*, pp. 477–481, 17th International Conference on Data Engineering, April 2001.

[11] Brin, S. and Page, L., "The anatomy of a large-scale hypertextual web search engine," tech. rep., Stanford University, 2003.

[12] Burners-Lee, T., "Information management: A proposal," in *CERN*, CERN, May 1989.

[13] Card, S. K., Moran, T. P., and Newell, A., "The model human processor — an engineering model of human performance," 1986.

[14] Chandy, K. M., Dimitrov, B., Le, H., Mandleson, J., Richardson, M., Rifkin, A., Sivilotti, P. A. G., Tanaka, W., and Weisman, L., "A world-wide distributed system using java and the internet," in *In Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*, 1996.

[15] Chandy, K. M., Aydemir, B. E., Karpilovsky, E. M., and Zimmermann, D., "Event-driven architectures for distributed crisis management," 15th IASTED International Conference on Parallel and Distributed Computing and Systems, November 2003.

[16] Decker, S., Melnik, S., andD D. Fensel, F. H., Klein, M. C. A., Broekstra, J., Erdmann, M., and Horrocks, I., "The semantic web: The roles of xml and rdf," in *IEEE Internet Computing*, vol. 4, pp. 63–74, IEEE, 2000.

[17] Fensel, D., "The semantic web and its languages."

[18] Hiler, J., "Google time bomb." http://www.microcontentnews.com/.

[19] Huang, J., Black, A., Walpole, J., and Pu, C., "Infopipes - an abstraction for information flow," (Budapest, Hungary), In Proceedings of the ECOOP Workshop on The Next 700 Distributed Object Systems, August 2001.

[20] Kobayashi, M. and Takeda, K., "Information retrieval on the web: Selected topics," tech. rep., Tokyo Research Laboratory, IBM, 1999.

[21] Modahl, M., Bagrak, I., Wolenetz, M., Hutto, P., and Ramachandran, U.

[22] Nardi, B., *Context and Consciousness: Activity Theory and Human-Computer Interaction*, ch. Studying Context: A Comparison of Activity Theory, Situated Action Models and Distributed Cognition. Cambridge, MIT Press, 1996.

[23] Roy, J. and Ramanujan, A., "Understanding web services," in *IT Professional*, vol. 3, pp. 69–73, November 2001.

[24] Singh, R., Li, Z., Kim, P., and Jain, R., "Event-based modeling and processing of digital media," First International Workshop on Computer Vision Meets Databases, 2004.

[25] Sousa, J. and Garlan, D., "Aura: an architectural framework for user mobility in ubiquitous computing environments," pp. 29–43, Software Architecture: System Design, Development, and Maintenance (Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture), Kluwer Academic Publishers, August 2002.

[26] Tilley, S., Gerdes, J., Hamilton, T., Shihong, H., Muller, H., and Wong, K., "Adoption challenges in migrating to web services," pp. 21–29, 4th International Conference on Web Site Evolution, October 2002.

[27] Wang, Y. and Wang, Y., "Cognitive models of the brain," in *First International Proceedings on Cognitive Informatics*, Cognitive Informatics, August 2002.

VITA

Derik Pack was born in Aiken, South Carolina. He received his bachelors of Engineering from the University of South Carolina in Computer Engineering. He is a Masters Student at the Georgia Institute of Technology where Ramesh Jain is his advisor. The focus of his research is the application of event models to web services and architectures. Once he has received his Masters he will begin work for Space and Naval Warfare in Charleston, SC.