Event-based Infrastructure for Reconciling Distributed Annotation Records

Ahmet Fatih Mustacoglu

Submitted to the faculty of the University Graduate School
in partial fulfillment of the requirements
for the degree
Doctor of Philosophy
in the Department of Computer Science,
Indiana University
July 2008

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Doctoral Committee

_____
Dr. Geoffrey C. Fox (Principal Advisor)

_____
Dr. Steven D. Johnson

_____
Dr. Andrew Lumsdaine

_____
Dr. Kay Connelly

July 31, 2008

Abstract

Information is spread all over the Web in various locations including centralized repositories, web servers and user desktops. Centralized repositories represent the old fashion techniques for resource sharing, whereas completely decentralized systems such as P2P systems allow users to share information without depending on a third party repository. The necessities to find and share information led to development of emergent Web 2.0 applications. These new Web 2.0 applications such as social bookmarking tools introduce a new way of sharing information rather than the old fashion and P2P systems do. Social bookmarking tools address the challenging problems of finding and sharing information among small groups, teams and communities. Various types of social bookmarking tools developed their own systems to support different kind of resources. Flickr, for example, allows the tagging and the sharing of photos, del.icio.us the tagging and the sharing of bookmarks, Bibsonomy, CiteULike and Connotea the tagging and the sharing of scholarly publications, YouTube the tagging and the sharing of video, and 43Things the tagging and the sharing of goals in private life. Social bookmarking tools for sharing of scholarly publications among these solutions are not interoperable with each other. Furthermore, they have limitations for representing the complete metadata of scientific documents and providing timestamp information for updated records.

In this dissertation, we present service enabled Event-based Infrastructure to provide an efficient, scalable, flexible and modular architecture to represent and reconcile metadata of scholarly publications coming from various sources. The system utilizes Event-based Infrastructure and adopts an optimistic replication approach to represent the

content of scientific documents located at several annotation tools consistent with each other with the added metadata fields and capabilities. We also present an empirical evaluation of the system to demonstrate applicability of this architecture to handle with the issues that exist in the annotation tools for scholarly publications.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

One of the major challenges that people facing with is to remember and access information that they have found earlier and thought could be useful for them later. Probably the most common approach to re-finding information on the web is to use personal bookmarks provided by several web browsers. For instance, Mozilla Firefox browser supports the creation of collections of URLs. Furthermore, URLs can be annotated by using keywords or free-form text. These collections can also be sorted based on a various things such as keyword, last visited, location or time. People created bookmarks depend on their personal interests in the information and quality of the resource, possibility of future use, current necessities as explained in [1].

Information is spread all over the Web in various locations including centralised repositories, web servers and user desktops. Centralised repositories represent the old

fashion techniques for resource sharing, whereas completely decentralised systems such as P2P systems allow users to share information without depending on a third party repository. The necessities to find and share information led to development of emergent Web 2.0 applications. These new Web 2.0 applications such as social bookmarking tools introduce a new way of sharing information rather than the old fashion and P2P systems do.

Social bookmarking tools address the challenging problems of finding and sharing information among small groups, teams and communities. Various types of social bookmarking tools developed their own systems to support different kind of resources. Flickr, for example, allows the tagging and the sharing of photos, del.icio.us the tagging and the sharing of bookmarks, Bibsonomy, CiteULike and Connotea the tagging and the sharing of scholarly publications, YouTube the tagging and the sharing of video, and 43Things the tagging and the sharing of goals in private life. Social bookmarking tools for sharing of scholarly publications among these solutions are not interoperable with each other and they have limitations for representing the complete metadata of scientific documents and providing timestamp information for updated records.

There are several common features for social bookmarking systems. First of all, these tools provide their users with ability to create their personal bookmarks and share them with other users instantly. These personal bookmarks are stored centrally in these systems and can be accessible from any computer that has an internet connection. Second, these systems enable entering personal keywords called tags explicitly by the user for each bookmark. Using tags for the resources allows users to organize and display their collections in a meaningful way. Furthermore, assigning multiple keywords for a

bookmark make it belongs to multiple categories. The final common feature of social bookmarking tools is the social way of their use. The collection of bookmarks created by users is also visible to other users. For instance, when a user name is clicked on, then the collection of bookmarks for that user is viewable to other users. Similar transparency is also valid for tags. So, one can retrieve similar resources that fall into same interest of other users by clicking on an interested tag.

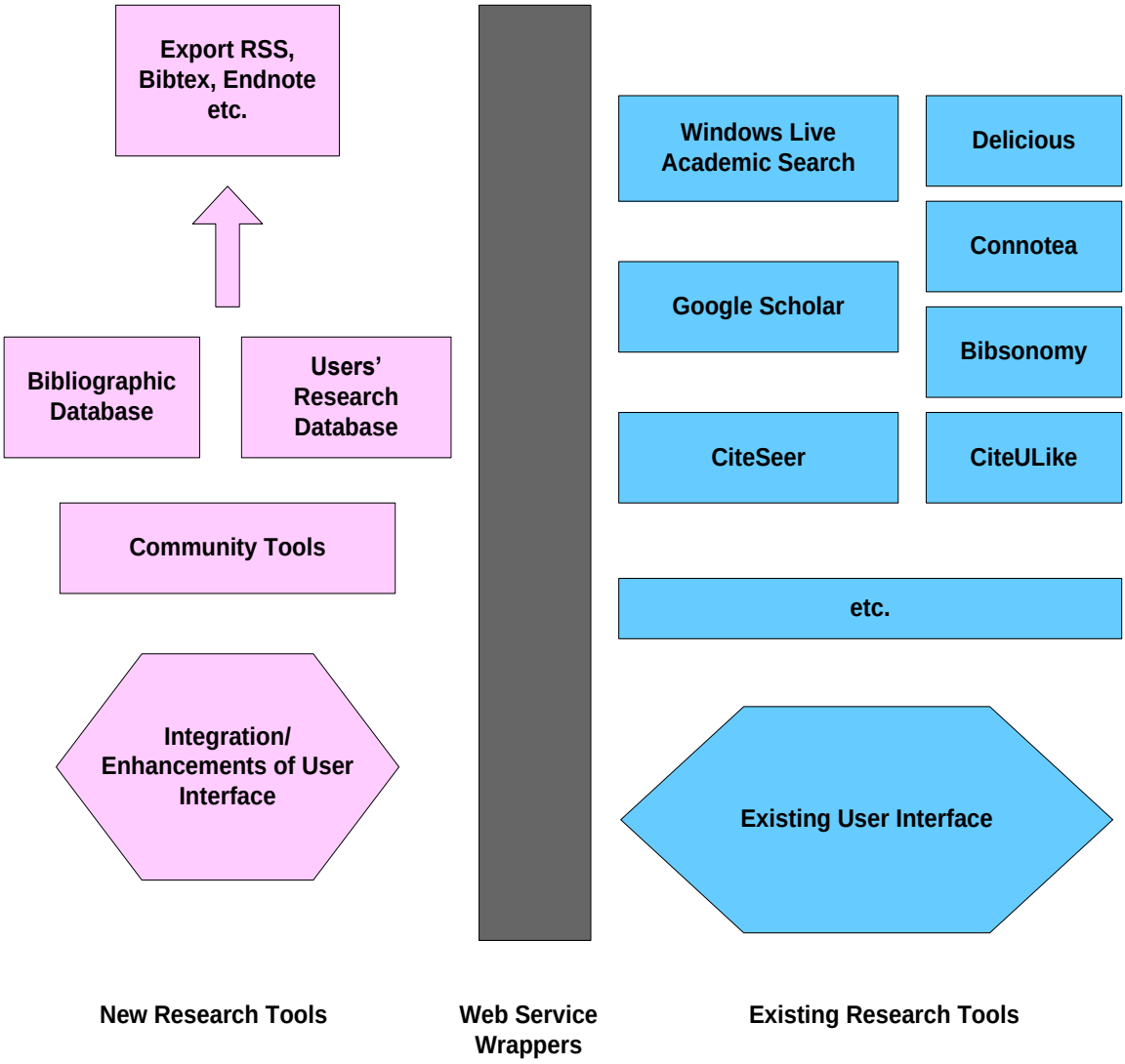

Figure 1-1: Research Tools with added capabilities for Sharing and Managing Scientific Documents

Search tools have been developing rapidly and supporting the collection of documents and metadata about scientific documents. The most famous of the search tools are the Google Scholar and Windows Live Academic Search. Google Scholar, for instance, provides various types of metadata about scholarly publications such as number of cited, conference name etc.

Figure 1-1 illustrates a model of building a system hierarchy where search tools and existing services of social bookmarking tools can be used with added capabilities to collect and manage metadata and data for scientific content. Our goal is to define the practical extent of existing annotation tools for scholarly publications based on information retrieveal and management in a consistent way.

1.1 Motivation

As the web-based social bookmarking services have gained popularity, an emerging need has appeared for methodologies to retrieve, represent, share and manage information that are stored in these annotation tools for scholarly publications. As these services enable storing, tagging and sharing documents, another emerging need has also appeared for supporting these tools by using their existing services via Web Service wrappers with added capabilities. In this thesis, we are particularly interested in investigating, managing, sharing and reconciling scholarly publications that are stored in several social bookmarking tools in a Service Oriented Architecture.

We identify the following limitations of current annotation tools for supporting the management and sharing of scientific documents.

First, there has been increasing number of annotation tools, each having their own structure and design, their own interface, their own format of their holdings. Even though

these tools provide common features to their users such as tagging, storing and sharing metadata and data, they do not have complete metadata support to represent the whole content of a scientific document. Because of this, users are forced to save their interested publications or documents into several annotation tools.

Second, annotation tools are lack of support for communication with each other to exchange or share information. Hence, users of these systems suffer from having the same documents in several tools and not being able to form a whole document by combining the pieces from various annotation tools.

Third, these tools are also missing services for uploading, extracting and importing documents from/to various locations. Users of these systems will not have any choice to extract the content of their documents stored at annotation tools into a specified repository or to store their metadata and data from a specified repository into these annotation tools. As a result of these, users of these systems can keep their documents on a shared place provided by these tools, and use their services to share, store new documents or modify existing documents, and tag their documents by using these tools' user interfaces.

Fourth, annotation tools do not provide timestamp information for their updated records. Users of these tools and services can share, update or modify content of documents without timestamp information. Because of this, the same documents can be updated inconsistently with unknown precise timestamps and spread around in existing annotation tools with different versions resulting in inconsistencies. In order to keep all replicas of the same document at several annotation tools, there is a need for reconciliation of distributed annotation records located at various annotation tools.

Fifth, existing annotation tools to store, tag, share scientific documents, and find similar groups or documents can be supported by using these tools' existing services in a uniform interface with additional tools (such as Google Scholar, CiteSeer, and Windows Live Academic etc.) and capabilities for collecting, sharing, tagging and managing scholarly publications for scientific research. To do so, there is a need for an infrastructure to represent content of scientific documents and their metadata coming from various sources. This infrastructure should also enable keeping track of changes to documents and metadata.

1.2 Statement of research problems

In this thesis, we mainly focused on investigating a novel approach of building consistent, Event-based Infrastructure to reconcile multiple sources of publications coming from various sources. In order to build such framework, we particularly identify the following research questions.

1. Can we implement an infrastructure that handles data and metadata coming from various sources in Service Oriented Architecture? Can this infrastructure unify and federate various existing online annotation tools for publications, which stores replicas of the same documents, and use their services? What is the efficient and flexible data model for such framework?
2. How can we support a flexible architecture that allows users to easily track documents?

3. How can we provide a consistency mechanism between the online replicated documents stored at annotation tools for scholarly publications and document located on a central server?
4. How can we achieve an information management architecture that can provide more metadata support than the current annotation tools do for scholarly publications?
5. Can we support communication between annotation tools for scholarly publications?
6. How can we provide users with ability to access previous versions of an updated document? Can we allow users to retrieve and apply other users' updates for a same document? What is the flexible update model?
7. Does event-based approach scales very well?
8. Can we support services for extracting data and metadata from these annotation tools into a specified repository? Moreover, can we support services for uploading data and metadata from a repository to annotation tools?

1.3 Why Event-Based Infrastructure and Consistency Framework for Distributed Annotation Records?

There are increasing numbers of annotation tools for scholarly publications and we are not sure that which one will survive in the future. So, one can not trust to keep his/her research works in only one tool. Replication of the documents in other annotation tools can be seen as a solution to keep documents safely at first glance, however none of the tools provide complete metadata support for the documents resulting in having

various incomplete versions of documents in several annotation tools. Another limitation is the support for the timestamp information in these annotation tools. They do not provide timestamp information for the updated entries. Furthermore, these tools are lack of interoperability to exchange data and metadata between each other. As a result, why can not we use these existing annotation tools and their services with added extra capabilities to provide a framework to manage documents, which are coming from various sources and consistent with their copies? Apperantly, our proposed Event-based Infrastructure (EBI) and Consistency Framework for Distributed Annotation Records (CFDAR) offer a solution to manage documents in a consistent manner.

Since EBI is an event-based system, it only stores the changes that happen within the system in a database. This reduces any additional computation to check the current status of a document. Having an event-based model also provides easy and flexible document tracking and navigation through the histories of documents. We never loose a version of a document, and each modification is kept as an event in our proposed EBI. So, we can easily rollback by undoing changes to modify the current content of a document to a previous version of it. One major drawback of keeping every change as an event is to have enough space. Another limitation is to necessary time to spend to process events to build a current version of a document. But, today's computers are fast, hard drive and memory are cheap. So, these limitations can easily be handled by using powerful computers with huge hard drive capacity and fast memories.

Having stored documents and keeping the modifications to them as events in a database with additional metadata support and capabilities allow us to have complete version of a document in a safe place with various abilities to manage them. Furthermore,

timestamp information for every change for each document is also provided by EBI for tracking the changes to documents in an easy and flexible way.

CFDAR provides a consistent view of documents between their replicas at several annotation tools and a complete version of documents stored in a central MySQL system database. To do so, CFDAR adopts optimistic replication approach and utilizes several services, which communicate with annotation tools and the database, for getting and distributing updates between the complete version of a document and its replicas.

Web Services constructs loosely coupled systems to enhance interoperability between applications running on different platforms. Similarly, we can benefit from Web Services to leverage the interoperability between the annotation tools to provide them with ability to communicate with each other to exchange data and metadata.

1.4 Thesis Contributions

The main contribution of this thesis is unifying and federating major annotation tools by proposing an Event-based Infrastructure and Consistency Framework for reconciling Distributed Annotation Records coming from various sources in a consistent manner. The implications of this thesis include, but are not limited to:

- Proposing an infrastructure for handling data and metadata coming from various sources in a flexible manner. An example implementation of the proposed infrastructure is presented to manage documents coming from different sources. This approach introduces the Event-based Infrastructure discussed in CHAPTER 3 and has been implemented and tested in IDIOM prototype system [2-4] discussed in CHAPTER 5.

- Proposing a novel framework for unifying and federating various online annotation tools, which keep the replication of same documents, and using their existing services to benefit from them [2, 5]. This thesis introduces a common data model and communication language to provide a common platform where integrated annotation tools can interoperate and exchange information. With this approach we aim to enable different annotation tools with different capabilities to communicate with each other and utilize each other's services.
- Proposing a novel framework for maintaining eventual consistency between the central server, where the primary copy of a document resides, and the annotation tools, where the replication of a document is stored [6]. Details of the consistency framework are discussed in CHAPTER 4.
- Proposing an update model for updated documents to provide efficient and flexible modifications of metadata fields of a document discussed in detail in Section 3.5.
- Identifying the key factors and design issues that affect the management of documents coming from several sources. This framework provides a more metadata support for publications for better representing the documents than the major annotation tools do.
- Implementation of the proposed Event-based Infrastructure and Consistency Framework for Distributed Annotation Records System Software and its user interfaces [2] discussed in CHAPTER 5.

- Performing performance and scalability measurements to investigate the implementation of the prototype system throughly discussed in CHAPTER 6.

1.5 Methodology

To evaluate our architecture, we chose Apache Axis 1.2 [7] version to deploy Web Services and Apache Tomcat [8] is used as a servlet container. User interfaces are developed in Java Server Pages (JSP) in prototype system described in detail in CHAPTER 5. We have integrated Connotea, Delicious and Citeulike annotation tools into our prototype system as replications of documents that are located on a central server with additional metadata. To maintain consistency among those annotation tools and central server, we have applied our Event-based Infrastructure (EBI) and our Consistency Framework for Distributed Annotation Records (CFDAR) that is running in the background all the time. Our framework is developed and deployed in an open environment.

We performed several experiments by modifying the input parameters to figure out the behavior of EBI and CFDAR. EBI and CFDAR are able to handle multiple clients' requests concurrently with the database and the memory utilizations. Furthermore, we have investigated latency metric for the major operations of EBI and CFDAR.

Java 2 Standard Edition compiler with version 1.5.0_12 is utilized. Java is a platform independent object oriented language from Sun Microsystems [9]. Java is preferred language for Web Service technologies due to its platform independence. As a result, we have also selected Java as our programming language to benefit the technologies that are already developed for Web Services.

1.6 Thesis Roadmap

We have presented a general introduction of the proposed research in this chapter. First, the limitations in existing online annotation tools for scholarly publications, which lead into the proposed research, were discussed in detail. Second, the statement of research problems is given. Third, we have explained why we apply Event-based Infrastructure in this research. Fourth, we have provided the contributions of the thesis. Finally, we have explained our methodology in this thesis.

The organization of the rest of the thesis is as follows. CHAPTER 2 reviews the background information and the underlying technologies. It provides a survey of event systems and information about consistency maintanence. Several technologies are also explored in the second half of the chapter.

CHAPTER 3 presents Event-based Infrastructure in detail. The big picture is given at the beginning of the chapter, and it displays the general idea and principles of the proposed infrastructure. The modules of the proposed infrastructure are explored in the remainder of the chapter.

CHAPTER 4 describes the Consistency Framework for Distributed Annotation Records (CFDAR) in detail. This chapter explores the design overview, consistency criteria and CFDAR in detail. The modules are described in the remainder of the chapter.

We discussed a prototype system in CHAPTER 5 to demostrate the effectiveness and applicability of Event-based Infrastructure and Consistency Framework for Distributed Annotation Records. The prototype system is then subjected to several tests, which are analyzed to help clarify the key features of this thesis.

CHAPTER 6 analyzes the performance evaluation of the Event-based Infrastructure and Consistency Framework for Distributed Annotation Records. It presents bencmarking on performance, and scalability of the system. Finally, in CHAPTER 7, we present the thesis summary, answers to research questions and outline several areas for future research directions.

CHAPTER 2

BACKGROUND AND SURVEY OF TECHNOLOGIES

In this chapter, we have investigated the related work on relevant concepts covered in this thesis, summarize several well-known projects in the research community that are also closely related to our work and introduced major technologies applied in the thesis project. We will discuss Web 2.0 and Annotation Tools, Event Systems, Consistency Maintenance, Related Projects, and Technologies in the following subsections.

2.1 Web 2.0 and Annotation Tools

In recent years there has been a rapid development of tools and services aimed at fostering online collaboration and sharing between users and communities [10]. Blogs

(blogger.com, Google Blog) [11], Wikis (Wikipedia, WikiWikiWeb, Wikitravel) [12, 13], Social Networking Tools (MySpace [14], LinkedIn [15]), Social Bookmarking Tools (del.icio.us [16], Flickr [17], YouTube [18]), Syndication Feed Aggregators (Netvibes [19], YourLiveWire [20]) and other related tools are quickly being embraced by an expanding user base. The term "Web 2.0" is now a widely accepted term representing this wave of new Web-based tools and the belief that they indicate a qualitative change in today's Web.

This change is also apparent in the domain of scientific research, with the recent creation of a number of online tools that enable the annotation and sharing of scientific content, such as CiteULike [21], Connotea [22] [23], and Bibsonomy [24]. Perhaps, the best known annotation (or, social bookmarking) web site is del.icio.us (henceforth referred to as Delicious) [16], a tool designed to enable the annotation and sharing of URLs. A number of other annotation tools that support collaborative tagging [25-28] are now in widespread use; they support annotation and sharing of a variety of resources, such as photos (Flickr), videos (YouTube), books (LibraryThing [29]) and goals (43things [30]). In particular, there are several online tools specializing in the annotation of scholarly publications, including Connotea, CiteULike, and Bibsonomy. The core service offered by these annotation tools is the capability that allows users to quickly annotate their favorite resources (URLs, photos, or citations) using a small number of tags (keywords) and to share their tagged content with other users. Tagging represents a significant shift in the *metadata creation* methodology. Traditionally, metadata creation has been handled by: (a) specialized professionals working with complex categorization schemes; or (b) the authors of scholarly content. Both of these methods suffer from

various problems [31]. Among the cited shortcomings of professional metadata creation are the complexity and the lack of scalability of cataloguing systems, especially when applied to the vast amount of data in today's Web. Author metadata creation is vulnerable to inadequate or purposefully inaccurate descriptions by authors. The new approach of metadata creation, namely *tagging*, puts the task of metadata creation in the hands of general users. This practice of collaborative categorization (which is now commonly referred to as *folksonomy* [31, 32]) aims to harness the collective intelligence of a large number of people. It has met with widespread acceptance by the Web users, as shown by the sharp increase in the number of subscribers to such tools. Recently, there have been preliminary attempts to look into the cognitive underpinnings of the popularity of tagging [33] and some dynamic discussions about the bottom-up tagging versus top-down categorization trade-off [34, 35]. While tagging remains a new practice whose long-term benefits are not yet well-understood, some of its advantages and disadvantages have already been pointed out [36]. Among the benefits of tagging are: (a) the ease of use and access of the tagging tools; (b) the ease of discovering new content; (c) the support for the creation of niche communities. The shortcomings include: (i) the lack of a standard set of keywords; (ii) the difficulty of dealing with misspelling errors, synonyms, and acronyms, which are commonly found in tagging; (iii) the difficulty of inferring hierarchical relationships between tags (i.e., creating taxonomy). Each social bookmarking tool can be described in terms of: (a) A model of data and metadata adopted by the tool; (b) A user interface that allows users and groups to subscribe to the service, manage their tagged content, share it with other users, and discover new content; (c) An input/output interface that allows the data and metadata to be exported to various formats

or applications, and enables programmatic interaction with the system. An overview of these features for the case of Delicious is given next. Table 2-1 summarizes the features of three other tools (CiteULike, Connotea, and Bibsonomy) in addition to Delicious.

a) Data and Metadata: There are two main *data* objects handled by Delicious: *users* and *URLs*. Anyone can register by creating a user name and a password. Users maintain lists of (annotated) URLs which they can share with other users. In addition to these two data objects, there are several types of *metadata*:

User network: Users self-organize into a network through a simple process whereby any user A can designate any other user B as being "*in her network*". In this case, user A is said to be a *fan* of B. This process leads to the creation of a directed graph whose nodes denote users and where an arc (*u*, *v*) means that user *v* is in user *u*'s network (or, that *u* is a fan of *v*).

Bookmarks: Users can add annotations to their favorite URLs, thereby expanding URLs into bookmarks. There are three different types of annotation in Delicious: *descriptions*, *notes*, and *tags*. The *description* of a URL is the *title* of the web page addressed by that URL (i.e., the text between <title> </title> HTML tags in the source code of that page). *Notes* and *tags* are user-defined annotations. Notes are expressions or sentences that describe the content of a URL. Tags are single-word, freely chosen descriptors of a URL and represent the most widely used type of annotation. A user can assign as many tags as she likes to a URL and can even rename or delete these tags later. There are no restrictions in choosing tags (except that a tag can't contain a space); thus, a tag can be an English word, an abbreviation, an acronym, a sequence of non-alphabetic symbols, etc. A user can group her tags into *bundles*. A bundle should be composed of a

set of tags which are somehow related (the name of a bundle should reflect the way in which its tags are related). A tag may belong to several bundles.

b) User Interface: The user interface of Delicious provides a number of ways in which the users can add, share, and discover bookmarks.

Adding bookmarks: Each user maintains a list of favorite bookmarks. This list can be populated in two ways: (i) by installing a *bookmarklet*—a button which when clicked triggers the execution of a piece of Javascript code—in the browser and clicking it while visiting a web page that is to be bookmarked; (ii) by manually creating bookmarks while logged into the system.

Sharing bookmarks: A simple way in which a user can share her bookmarks is by emailing the URL of the web page containing her favorite bookmarks to the people she would like to share her bookmarks with (this URL has the format http://del.icio.us/<uname>). A member can share a bookmark with a specific other member by tagging the bookmark with the "*for: uname*" tag; this bookmark will appear in the "*links for you*" page of the other member.

Discovering new bookmarks: In addition to discovering new web pages through standard methods, such as search engines and topic directories, one can also discover interesting pages by *browsing* or *searching* the data and metadata stored in Delicious. Currently, Delicious provides support for the easy browsing of the *recent* and *popular* bookmarks and tags, the bookmarks and tags of a *particular user*, the bookmarks tagged with a *particular tag* or the ones with a *certain media type*. Another way of discovering new bookmarks is to *subscribe* to one or more tags of interest. After you subscribe to a set of tags, Delicious keeps track of all bookmarks subsequently tagged with those tags

and shows them to you under the "subscription" page. A user can see the favorite bookmarks of all users in her network. Finally, it is also possible to search for bookmarks by keyword.

By default all information in Delicious is *publicly viewable*. However, it is possible for a user to declare one or more bookmarks, or her network, as private.

c) Input/Output Interface: There are several ways in which a program can exchange data with Delicious:

- The Delicious API is, as of this writing, in the initial phase of development. Currently, it provides methods for (i) checking the time when a user last posted a bookmark; (ii) obtaining the list of tags of a user, and renaming them; (iii) obtaining the list of bookmarks of a user, modifying, or deleting them and adding new bookmarks; (iv) obtaining the bundles (i.e., tag sets) of a user, deleting bundles, or creating new ones. All communication with the API is done over HTTPS. A delay between queries of at least 1s is required by the system.
- JSON (JavaScript Object Notation) [37] feeds are available for: bookmarks, tags, network, and fans.
- RSS [38] feeds are available on most pages within Delicious; no RSS feed is allowed to be polled more frequently than once every 30 minutes.

Table 2-1: A summary of the features of Delicious, CiteULike, Connotea and Bibsonomy

	Delicious	**CiteULike**	**Connotea**	**Bibsonomy**
Data Model	>> users >> general URLs	>> users >> groups >> citations	>> users >> groups >> citations	>> users >> groups >> general URLs or citations
Metadata	>> user network >> other networks (users-tags, bookmarks-tags, users-bookmarks) >> descriptions of URLs >> tags, bundles of tags >> notes on URLs	>> authors >> tags >> notes	>> tags >> descriptions >> comments >> geographical metadata (by GoogleEarth) >> tag notes (i.e., tag annotations)	>> network of "friend" users >> tags >> descriptions >> tag relations (subtag, supertag)
User Interface	>> adding bookmarks • bookmarklet • importing from favorites stored in browser • manual >> modifying bookmarks • add/delete/rename all annotations • delete bookmarks >> sharing bookmarks • email bookmark page's URL • tag with "for: uname" >> discovering bookmarks • browse (hot now, recent, popular, specific tag, specific media type, history) • for a particular tag, see related tags, active users • for a particular bookmark, see common tags, related bookmarks, posting history • see bookmarks of all users in my network • "links for you" page • subscribe to specific tags • search: - by default search tags, notes, descriptions - may search only tags - operators: AND, OR, -, NOT, XOR	>> adding bookmarks • bookmarklet (only for supported publisher sites) • manual • import from Bibtex >> modifying bookmarks • can add/delete/rename allcitations fields • can delete citations >> sharing bookmarks • email bookmark page's URL • automatically exported to "Everyone's library" >> discovering bookmarks • browse - everyone's library - everyone's tags - a specific tag - a specific author - a specific user - a specific group • create a watchlist of tags, users, groups • search by keyword one of: title, author surname, abstract, journal name, tag	>> adding bookmarks • bookmarklet (only for supported publisher sites) • copy another user's bookmarks • manual (supports DOIs) • import from local file (RIS, Bibtex, Endnote) >> modifying bookmarks • add/delete/rename all citation fields • delete citations >> sharing bookmarks • email bookmark page's URL >> discovering bookmarks • browse - popular bookmarks - popular tags - a specific tag - a specific user - a specific group - related tags • search can choose to search one of: my library, user, tags, all)	>> adding bookmarks • bookmarklet • copy another user's bookmarks • manual • import from Bibtex snippet >> modifying bookmarks • add/delete/rename all citation fields • delete citations >> sharing bookmarks • email bookmark page's URL >> discovering bookmarks • browse - popular bookmarks - popular tags - a specific tag - a specific user - a specific group - related tags - suggested tags • search can choose to search a user's metadata or all users' metadata
I/O Interface	>> API • support for tags, bundles, bookmarks, posting times • over HTTPS • delay between queries > 1s >> JSON feeds • bookmarks, tags, user network, user fans >> RSS feeds • available for most pages • delay between polls > 30 min	>> RSS feeds >> Export to Endnote, Bibtex	>> API • over HTTP • retrieve list of bookmarks • retrieve list of posts • retrieve list of tags • create a new post • edit existing post • remove existing post >> RSS feeds >> Export to RIS, Endnote, Bibtex, MODS	>> RSS feeds >> SWRC feeds >> Export to Endnote, Bibtex

- **Discussion:** While we expect that annotation tools will constantly improve, it seems unlikely that all of them will "prosper". This uncertainty will clearly inhibit adoption; therefore, we adopt a philosophy that is different from the one that specializes Delicious to scientific content in Connotea and CiteULike. We do not intend to replace any of these systems in our research but rather add to them by building tools that add new capabilities. We will achieve this by building wrappers (constructed as Web services) which allow us to both extract information from these tools and to store information in them in our thesis research. While doing that, inconsistency issues rise up due to updates in records without the time stamp information for the updated entries in these tools. To handle inconsistencies that might occur among entries we propose our Consistency Framework for Distributed Annotation Records (CFDAR) described in [6].

2.1.1 Related Projects

Major related projects in social bookmarking and the ones that use bookmarking tools as a base for their design are summarized in the following sub-sections.

2.1.1.1 Connotea

Connotea is an open source free online reference management and social bookmarking service for scientific research community [23]. It is developed by Nature Publishing Group [39]. Connotea has currently huge and increasing number of users. Connotea project is inspired by the general web linking management system del.icio.us [16] to fill the gap in the field of scholarly reference management. Key features of Connotea project can be summarized as below:

- *Online storage of reference and bookmarks:* The current reference management systems rely on having a reference database stored to user's own computer locally, and modified via installed software. Having an online database has some advantages: (a) Allow resources to be available and accessible from any web-enabled computer; (b) It is easier to share references; and (c) Supporting other key features of Connotea system and direct linking of literature.
- *Simple, non-hierarchical organizing:* Reference data is not placed in folders or sub-folders; instead data can be viewed from the perspective of tags, users, or links. Tagging provides a grouping related documents and easy navigation of material without using nested hierarchical folder structure
- *Opening the list to others:* Connotea supports public and private bookmarking concepts to allow other users to see public documents or not to have access to private reference materials. Connotea can automatically discover and display connection between users based on the similar stored bookmarks.
- *Auto*-discovery of bibliographic information: Connotea can find and import the bibliographic information for any article or book. This eliminates the typing errors and reduces the amount of typing that users need to do.

The main futures of Connotea system that implements the concepts explained above are:

- *Bookmarklets:* Bookmarklets are the JavaScript code, and they can be integrated into browsers to provide users with custom functionality. One of the major Bookmarklet is the one that allows a user to save a webpage that he/she is currently viewing into his/her Connotea account.
- *Recognising URLs from common archives and importing bibliographic data:* Connotea has a built-in functionality to identify URLs if they belong to the set of URLs that Connotea recognizes. For example, if an added URL refer to a scholarly article, then Connotea stores the publication name, volume, issue number, publication date and list of articles (See comparison table to see available metadata field in Connotea in Table 3-1.
- *Tagging:* Tagging plays a crucial role in Connotea since users' referecences can be organized and grouped by user defined meaningful keywords called tags.
- *Comments:* Comments are the piece of personal data about documents. There also exist a bookmarklet plugin in Connotea to add a new comment about the current webpage that user viewing for an article or any document.
- *RSS:* The documents/bookmarks in Connotea can be navigated through the user, tag or the combination of user and tag. Every list of bookmarks in Connotea provides a RSS feed with subscribed users to notify them about newly added items.

2.1.1.2 BibSonomy

BibSonomy is a web-based social bookmark and publication sharing system [40]. The data model of publication documents is based on BIBTEX [41], famous literature management system for LATEX [42]. Bibsonomy can display bookmarks and BIBTEX based references at the same time depicted in Figure 2-1.

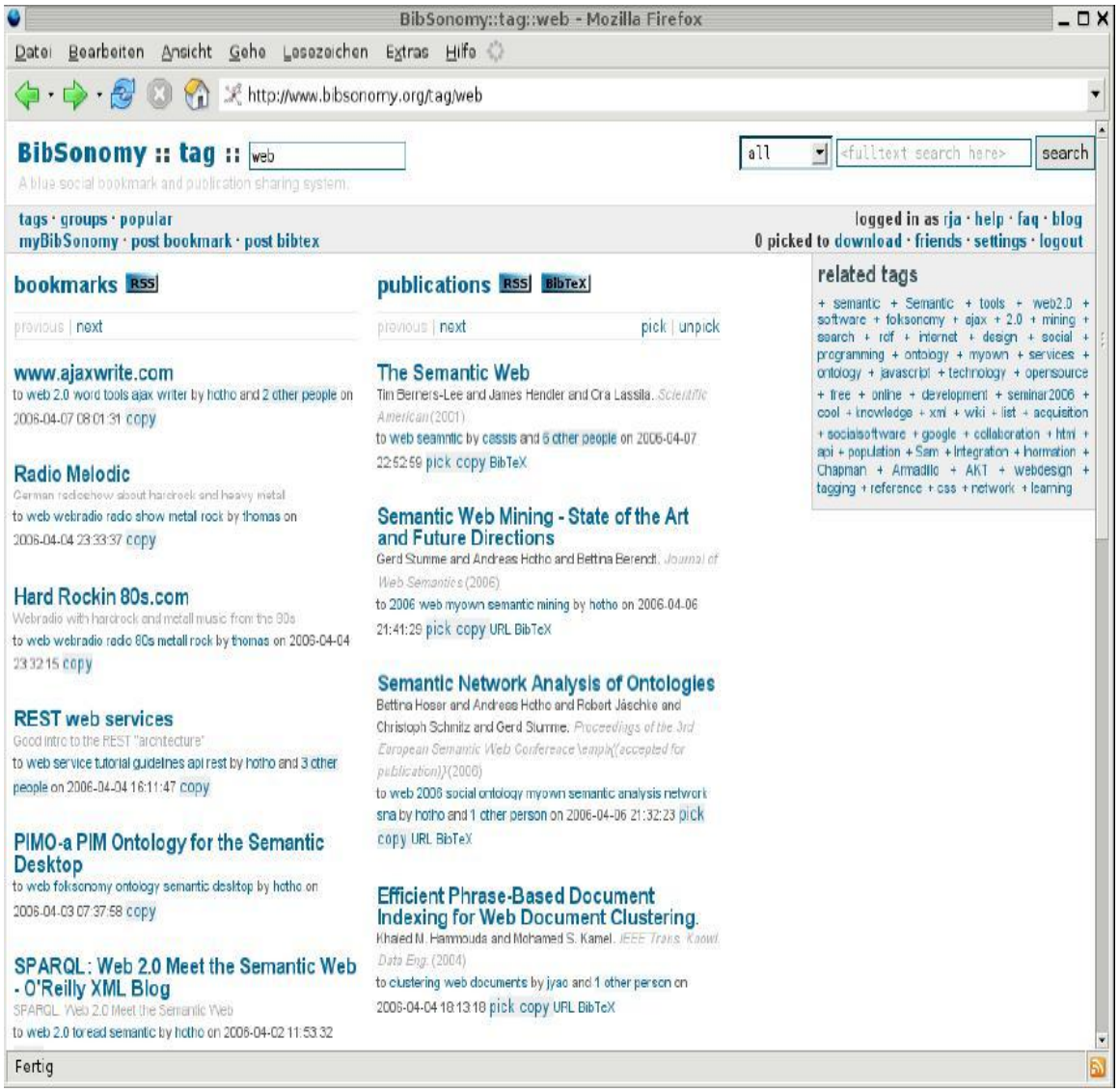

Figure 2-1: Bibsonomy User Interface showing Bookmarks and Publications Simultaneously (image taken from document [40])

The main architecture components of Bibsonomy are Apache Tomcat Servlet Container [8], Java Server Pages [43], Java Servlet Technology [44], and MySQL database.

Model View Controller (MVC) [45] programming paradigm is used for the development of Bibsonomy system to separate the logical handling of data from the presentation of the data. When there is a need for a new output format, it can be implemented as a JSP as a view of the model.

Bibsonomy relies on four major database tables: (a) A table for bookmarks posts; (b) A table for publication posts; (c) A table for tag assignmenst; and (d) A table for the relations.

The BibSonomy system provides its users with various services and main futures of the Bibsonomy system are: (a) Relations between tags (user specific binary relation explained in detail in Section 3 in [40]); (b) Duplicate detection (hashing BIBTEX fields by MD5 [46] message digest algorithm to detect duplicate publication entries); (c) Editing tags; (d) Importing resources; (e) Exporting resources; (f) Groups; and (g) Shopping basket, which allow picking desired publications to be available for export later in a shopping basket.

2.1.1.3 ShaRef

ShaRef is a reference management system with collaboration and lightweight knowledge management features [47-49]. ShaRef project is funded and carried out by Swiss Federal Institute of Technology (ETH) [50].

One of ShaRef's major goals is to manage references in a way that seamlessly integrates bibliographic and web references. This allows ShaRef to manage all references

uniformly and its knowledge management capabilities can be used for all types of references. It enables users to go from their references directly to the library via its OpenURL [51] service. ShaRef provides minimal but usefull support for knowledge management. It provides two concepts: (a) keywords, which are used for identifying references within ShaRef like tagging documents; and (b) cross-references, which connects references. ShaRef allows users to make generic cross-references such as creating annotations that refer to other references.

Sharef provides its user with ability to share their resources based on access control for users and groups. It also supports messasing to improve collaboration. So, users or member of groups can access to references via ShaRef interface. Furthermore, ShaRef allows users to publish references, and this enables the Web-based access to the published references. By integrating a user-defined XSLT program, any export format can be generated for references.

ShaRef has been design based on the XML data model and various XML technologies [52]. It is implemented by using the Java language for both platform and the client to be as independent as possible from any specific operating system. The Java rich client is client/server architecture and implemented by using Java RMI [53]. ShaRef can work in online or offline mode. When it is working online mode, it uses pure Java DBMS [54], whereas it uses a user's local hard drive in offline mode to store references. ShaRef also provides a Web-based user interface that enables accessing ShaRef references via a Web browser. The Web-based user interface does not offer the full services that Java client does.

2.1.1.4 ReMarkables

ReMarkables is a Web-based research collaboration support system that uses social bookmarking tools [55]. ReMarkables research project is developed at Nippon Institute of Technology [56], and available from http://comweb.nit.ac.jp/ReMarkables. It is built on existing social bookmarking tools and its main goal is to improve existing social bookmarking tools to provide efficient collaborative support system for scientific research communities. Members of scientific communities are mostly interested in research topics and expert groups in those topics. In ReMarkables system, it is easy to find research groups working on an interested topic via its topic bookmark services. Users of research communities can share their bookmarks with specific tags; communicate with each other using mailing lists and wiki pages associated with a topic bookmar list. ReMarkables system's requirement analysis is done based on Tropos methodology [57], and main functionalities of ReMarkables system are: (a) Retrieve online publications, search for topic bookmarks and other users' bookmarks; (b) Explore bookmarks; (c) Store personal bookmarks; (d) Export bibliography; (e) Manage topic mailing list; (f) Share topic bookmarks; (g) Share topic wiki page.

In Tropos methodology [57], the system's global architecture consists of subsystems, which are also called actors, and the subsystems are connected to each other via data and control flows (data dependencies). To provide the major functionalities of the ReMarkables system, new actors are introduced in the architecture of it and the main actor is called "Topic recommender". Topic recommender has three main functions: (a) Notify users via sending email when a new topic bookmark list is created from a search result for a specific topic; (b) Notify users via email when a new bookmark is added; (c)

Display related topics in the topic bookmark frame when a user retrieves online publications. Finally, the whole ReMarkables system services are managed and provided by the five major architecture components: (a) Bookmark Searcher; (b) Personal Bookmarks Manager; (c) Bibliographer; (d) Topic Recommender; and (e) Topic Bookmarks Manager.

2.2 Event Systems

In recent years, there has been an increasing amount of research focused on event based systems. Their main objective is to notify the necessary entities about the changes that occurred in the domain of interest. Today, event systems are needed and used in several areas such as graphical user interfaces, databases, web based applications, networking applications, distributed applications, publish-subscribe paradigm etc. Several tools have been developed for each of these areas to satisfy their needs, and NaradaBrokering [58-62] is an open-source messasing infrastructure, which implements the publish-subscribe paradigm, developed in Community Grids Lab at Indiana University [63].

There are two different approaches to the event definition. The first approach defines an event as it is an instantaneous atomic occurrence, so it is represented as a point in time [64-66]. Based on this approach, timestamps of event occurrences can be categorized in three different ways:

- Absolute time point: It consists of date and time
- Relative time points: It is defined relative to a particular position
- Virtual Clocks are explained in detail in [67], and uniquie timestamp values are assigned automatically to each event by the system.

The second approach defines an event as occurrence as an interval in time [68-71]. Based on this approach, a state change of an event can be specified within a specific interval and the interval can be represented in two ways:

- As relative, absolute, or virtual time points represent starting and ending point of an interval
- Event occurrences that represent the initial and ending points of an interval

So, first approach defines events as having no duration while the second approach defines events by having them particular duration. Most of the previous event system related works use the first approach in their event-based modeling and design.

- **Discussion:** In our research, we have choosen to use the first approach due to its suitability to our design of the event-based infrastructure. We assign a time stamp value to each minor or major event once they occur within the system as an absolute time point described in [3]. This time stamps values provide us with ability to sort events based on their occurrences and to use time stamp values for consistency maintanence described in detail in Section 4.4.

2.2.1 Event Representation

According to [72-74], events are represented as tuples. Since any state change of an event in a specific time point or an interval represents information, which is defined as a data structure with several attiributes. Events are constructed in the form of tuple structure and delivered to external entities that are listening the system for a particular state changes. The communication model for delivering events in the form of tuple structure to the external entities takes place in the form of messages. Message formats

varies based on the domain of each system. Messages in event system represent a tuple structure and tuples genericly composed of:

- Unique Event Id
- Event attributes that caries additional information about the event

The unique event id helps an event to be separated from other events and it is a mandatory field for event representation. Event attributes carries an extra information related to the event such as event type, event owner, etc.

Events are described as in the form of tuples with already built in abstract data types in previous work such as CORBA Event Notification Service [75, 76], Java AWT delegation Event Model [77], DOM [78] interfaces for tuple representation. In database programming, events are stored as a tuples in the form of record structures composing the event histories.

Every system has a response unit to the state changes coming from the environment to handle with the changes [73, 79]. Reactive applications depend on the data that describes the current state of their environment due to changes. Each application continuously checks any state changes happening in their environment to adopt the changes in their interest. The process of uninterrupted checking for detecting the state changes and retrieving the changes that represents the current environment is called monitoring the environment. Instead of monitoring the state changes, most of the systems prefer to be notified by the changes that happened in their domain of interest so that they do not need to monitor the state changes to reduce the computational works. Since monitoring the state changes requires an additional computational overhead, and at this point, event and event-based systems gets attention due to their nature. Use of event-

based systems provides applications with the state changes in their domain of interest in the form of messages without monitoring their environment. As a result, external systems do not need to spend any additional computation to retrieve the state changes. They can be notified by the event-based systems once a state change occurred.

In distributed event-based systems, multiple objects at different locations can be notified by events, which take place at any objects. To do so, they use publish-subscribe mechanism that allow an object to generate and propagate the type of events to all subscribed parties. Objects that are willing to receive updates from an object that has published its events subscribe to the type of events in their domain of interest. Different event types can point to different methods executed by the interested object. Notifications are the objects that represent events. Events and notifications can be used in various applications such as interactive applications, modifying a document, chat applications. Distributed event-based systems have two main characteristics [80]:

- *Heterogeneous:* When event-based systems are used for communication between distributed objects, different components that do not designed to work together can be interoperated. It is described in detail how event-based system can be used to interoperate different components on the internet [81].
- *Asynchronous:* Event generating objects send notifications to all objects that subscribe to them so that publisher do not need to synchronize with the subscriber objects. Project Mushroom described in detail in [82] is a distributed event-based system that supports collaborative work.

➢ **Discussion:** In our thesis, events have unique event ids, and we have distinguished our events as major and minor events and we have defined our events as a

time-stamped action on a digital document with additional information (described in detail in [3]). In our research, we have unified and federated heterogeneous annotation tools to communicate with each other via event-based infrastructure and Web service technology. We could not use publish-subscribe paradigm to disseminate updates since the integrated annotation tools do not support publish-subscribe mechanism. However, any application that require and support publish-subscribe mechanism, then broker address and topic can be defined in a property file of our proposed system to provide updates via publish-subscribe mechanism by connecting to the broker and subscribing to a topic. Finally, our update propagation falls into unicast communication technology.

2.2.2 Events Classification

Events are categorized as *Primitive Events* and *Composite Events*. The following sub-sections overview these two categories.

1) *Primitive Events*: They are the ones that are predefined within the system and originated from the publishers in the event notification service [83-85]. Some examples to primitive events: (a) Begin of Block (BOB) and End of Block (EOB) atomic primitive events in a multi-user DBMS system [86, 87]; (b) Predefined set of events in an embedded system like autonomous vehicle as in [88]. Computations are separated into several controllers and each controller can react based on the associated predefined set of events to prevent collisions; (c) Incoming requests for predefined database events such as insert, update, delete etc. in a centralized or distributed database system [89]; (d) Clock events generated using the system clock or distributed clocking scheme [90]; (e) Synchronous or asynchronous huge amount of data can be retrieved from the

sensors through an event notification service in a distributed sensor network system [91, 92].

Primitive events can be classified into four groups:

i. System related Primitive Events: They are the basic events that are defined and fired by the system such as events in graphical user interface (button press or release, etc.) [93, 94], database operations (update, delete, inset etc.), smart sensor systems (detection of temperature increase or decrease etc.) [95].

ii. User-defined Primitive Events: They are the events that are defined by users explicitly [96].

iii. Time related Primitive Events: These events are categorized into three types of events:

    a. Absolute Clock Events: These types of events are fired at an absolute time point instantenously.

    b. Relative Time Events: These events represent a unique time points that is defined by a given reference point and offset value for that point.

    c. Periodic Time Events: These events are defined with a reference point and a period. These events are fired by the system periodically from the defined reference point.

iv. Exceptions: They are the interrupts that occur in a system due to illegal operations. Exceptions can be seen in various types of applications,

especially in security systems and operating systems when a fault occurred in the system [97].

2) *Composite Events*: Primitive events are in wide use and they can satisfy the some application needs but, some of the applications may require more complex time pattern for their environment. At this point composite events rise up, and a composite event is a circumstance that represents a specific state change based on a pattern, which consists of a combination of the basic events. Some examples to primitive events: (a) To start a session when expected users finish their transactions in an active multi-user DBMS, a group administrator should track the system for a particular users and their transactions [98, 99]; (b) Tracking of the current traffic in a real time traffic control system to predict a traffic jam [100, 101]; (c) Providing necessary services for students and faculties in a Web-based education system [102, 103].

- **Discussion:** Events are particularly suited for object-sharing frameworks. They support heterogeneity. They enable us to meet users' requirement for highlevel awareness information during collaborations. They also fulfil users' requirement to inspect the histories of objects, and not just their current state. We have used event-based framework in our thesis research and our approach for event classification in our thesis defined in [3]. Our events (major and minor) are primitive events and they fall into system related primitive events since they are constructed by the system when an event occurred in the system. Each event has a unique event id, time stamp, event type information and payload data. That information is processed during document build-up process from

events to retrieve the latest or the desired version of a document and consistency maintenance period as explained in detail in CHAPTER 3 and CHAPTER 4.

2.2.3 Related Projects

2.2.3.1 JEDI

JEDI (Java Event-based Distributed Infrastructure) is an event-based, object-oriented infrastructure for the development of complex distributed systems [104]. JEDI infrastructure is based on the notion of active object (AO), which is "an autonomous computational unit performing an application-specific task" [104]. In JEDI, "Each active object has its own thread of control and interacts with other AOs by explicitly producing and consuming events" [104]. Events are messages and they do not include any information about their receipent. An event is ordered set of strings, which consists of event name and other parameter values. Events are generated by AO and sent to *event dispatcher* (ED). Event *subscription* and *unsubscribe* operations are provided for AO to allow them to show their interest for receiving or not receiving the interested events during their life cycle.

Main futures of JEDI can be summarized as below:

- *Event Patterns:* An event pattern is a set of ordered string, which represents a regular expression, in JEDI and AOs subscribe either a specific event or an event pattern. An event pattern consists of pattern name and pattern parameters. A pattern name is the first string, and the remaining strings are the pattern parameters in an event pattern.

- *Reactive Objects:* Reactive objects are the particular AOs that execute a standard loop by waiting for events that they subscribe to and process them in JEDI.
- *Distribution of the Event Dispatcher:* In JEDI, two versions of event dispatchers are supported: (a) Centralized; (b) Distributed. Centralized version consists of a single process, few AOs, running on local area network, and is designed to handle simple systems with exchanging limited number of events. So, centralized version can be a bottleneck for a distributed system. However, in the distributed version of the event dispatcher, the main goal is to support network intensive applications by exploiting a set of dispatching servers interconnected in a tree structure.
- *Preservation of Event Ordering:* In distributed systems, ordering of events ordering is a crucial issue, and none of the traditional communication mechanism used over the internet guarantees a total ordering of events due to variable latency. In JEDI, ordering of events is also crucial when distributed event dispatchers are needed to be used. As a result, in JEDI casual ordering of events are guaranteed [104].
- *Mobility:* Mobility is to be able to move running components of an application easily across to nodes of a network. In JEDI, mobile AOs are supported. So, AOs can disconnect from an event dispatcher and reconnect to another distributed event dispatcher. To provide mobility of AO, JEDI supports moveIn and moveOut operations [104].

2.2.3.2 NaradaBrokering

NaradaBrokering is an open-source event-brokering system based on the publish/subscribe paradigm, which allows distributed systems to communicate with each other by exchanging messages [58-62, 105, 106]. NaradaBrokering system has been developed at Community Grids Lab [63] at Indiana University and available from http://grids.ucs.indiana.edu/ptliupages/projects/narada.

Communication is asynchronous and events are central to NaradaBrokering system. Events encapsulate data in various levels, and they constitute the data flow in NaradaBrokering system. One of the main duties of NaradaBrokering system is to deal with efficient management of data flow.

NaradaBrokering system incorporates number of services: (a) Reliable delivery; (b) Ordered delivery; (c) Secure delivery of messages; (d) Access to globally synchronized timestamps; (e) Reduction of jitters. It also supports various communication protocols: (a) TCP; (b) UDP; (c) HTTP; (d) SSL; and (e) Parallel TCP.

NaradaBrokering also supports Java Message Service (JMS), JXTA to support peer-to-peer interactions, SOAP and several Web Service specifications including WS-Reliability, WS-Eventing, and WS-ReliableMessaging.

It has been used in various domains including collaborative applications, audio/video conferencing applications and GIS systems. Some example applications currently using NaradaBrokering are SERVOGrid [107], GlobalMMCS [108], the WEB-IS research work at the Florida State University and the University of Minnesota, and the Anabas system [109], which provides support for shared displays and online collaborative meeting software.

2.3 Consistency Maintanence

Consistency is an important issue in distributed systems. Consistency means that all copies of a same document meant to be the same. When one copy is updated, and then it must be ensured that all copies are updated as well [110].

According to [110], consistency models can be classified into two group: (a) Data-Centric Consistency Models; (b) Client-Centric Consistency Models. Details about these two models, update propagation and consistency protocols are given in the following sections respectively.

2.3.1 Data-Centric Consistency Models

A consistency model is an aggreement between processes and hosting environment, where data is stored. As long as processes obey the rules, the hosting environment promises to work correctly. A process that executes a read operation on a data item expects to get a value that is a result of the last write operation on the data item. However, in the absence of a global clock, it is difficult to say which write operation is the last one. So to maintain consistency in different ways, there are other data-centric consistency model definitions. Each data-centric consistency model has different restrictions on what a read operation can return on a data item. It is easy to implement and use consistency models with minor restrictions whereas it requires lots of effort to use consistency models with major restrictions. But the gain is different in each model since the one with major restrictions provide better results than the one with minor restrictions do [110]. More information on consistency models can be found in [111, 112]. Tanenbaum classifies data-centric consistency models into seven sub-categories:

- *Strict Consistency:* It is the most strict consistency model and it is defined by the following condition:

  > "Any read on a data item x returns a value corresponding to the result of the most recent write on x" [110].

  This model relies on absolute global time to order processes and all writes to a data item instantenously are visible to all processes. If a data item is changed, all read requests on that data item gets the new value, no matter how soon these requests are made, and which process are making a request and where these processes are located.

- *Linearizability and Sequential Consistency:* Sequential consistency model is a slightly weaker consistency model than strict consistency model. It is defined by the following condition:

  > "The result of any execution is the same as if the (read and write) operations by all processes on the data store were executed in some sequential order and the operations of each individual process appear in this sequence in the order specified by its program" [110].

  Linearizability consistency model is weaker than strict consistency model and stronger than sequential consistency model. A data store is linearizable as long as each operation has time-stamp value and the following condition is satisfied:

  > "The result of any execution is the same as if the (read and write) operations by all processes on the data store were executed in some sequential order and the operations of each individual process appear in this sequence in the

order specified by its program. In addition, if $ts_{OP1}(x) < ts_{OP2}(y)$, then operation OP1(x) should precede OP2(y) in this sequence" [110].

Linearizability is usually used in formal verification of concurrent algorithms [113]. If a data store holds necessary conditions for linearizable consistency, it also satisfies necessary conditions for sequential consistency. Preserving time-stamp ordered values brings additional cost to linerizability than sequential consistency [114].

- *Casual Consistency:* A casual consistency model is a weaker consistency model than sequential consistency model, and it distinguish events as casually related or not. So, if event X is caused or affected by an earlier event Y, than casual consistency requires that every process first get Y then get X. A data store said to be caually consistent if it satisfies the following condition:

  "Writes that are potentially casually related must be seen by all processes in the same order. Concurrent writes may be seen in a different order on different machines" [110].

- *FIFO Consistency:* FIFO consistency model is less strick than casual consistency model. FIFO consistency model requires the following condition to meet:

  "Writes done by a single process are seen by all other processes in the order in which they were issued, but writes from different processes may be seen in a different order by different processes" [110].

In distributed shared memory systems, FIFO consistency is called PRAM consistency, and is described in [115].

- *Weak Consistency:* Weak consistency is maintained by using synchronization variables to maintain consistency [116]. Weak consistency model has three characteristics as follow:

    1. “Accesses to synchronization variables associated with a data store, are sequentially consistent” [110].

    2. “No operation on a synchronization variable is allowed to be performed until all previous writes have completed everywhere” [110].

    3. “No read or write operation on data items are allowed to be performed until all operations to synchronization variables have been performed” [110].

- *Release Consistency:* This model provides two kinds of synchronization variables to distinguish a process whether enters or leaves a critical region. An *acquire* and a *release* operation notify a data store that a critical region is about to be entered or has just been exited [117]. A data store is release consistent if the following conditions are satisfied:

    1. “Before a read or write operation on shared data is performed, all previous acquires done by the process must have completed successfully” [110].
    2. “Before a release is allowed to be performed, all previous reads and writes done by the process must have been completed” [110].
    3. “Accesses to synchronization variables are FIFO consistent (sequential consistency is not required)” [110].

Lazy release consistency is a different implementation of release consistency and described in detail in [118].

- *Entry Consistency:* Entry consistency model has been designed to be used with critical sections [119] and it works in a similar way with release consistency model. However, unlike release consistency, in entry consistency each shared data has to be associated with some synchronization variable such as lock or barrier [110].

  A data store is entry consistent if it satisfies all the following requirements:

  1. "An acquire access of a synchronization variable is not allowed to perform with respect to a process until all updates to the guarded shared data have been performed with respect to that process" [110].
  2. "Before an exclusive mode access to a synchronization variable by a process is allowed to perform with respect to that process, no other process may hold the synchronization variable, not even in nonexclusive mode" [110].
  3. "After an exclusive mode access to a synchronization variable has been performed, any other process's next nonexclusive mode access to that synchronization variable may not be performed until it has performed with respect to that variable's owner" [110].

2.3.2 Client-Centric Consistency Models

In the previous section, we have overview and summarized data-centric consistency models that are all about providing a systemwide consistent view on a shared data. On the other hand, client-centric consistency models ensure the consistent view of

data from a client's perspective. They allow copies of a data to be inconsistent with each other as long as the consistency is maintained from a single client's point of view. Tanenbaum classifies client-centric consistency models into five sub-categories:

- *Eventual Consistency:* In eventual consistency, replicas are allowed to diverge and become inconsistent and it is guaranteed that the system can eventually converge to a consistent state. In this model, it is easy to solve write-write conflicts since; it is assumed that there are small numbers of processes that can perform an update operation.

  Eventual consistent data stores can perform well as long as clients always access the same replica, however problems may occur when different replicas are accessed. The problem can be solved via client-centric consistency that guarantees consistent view of a datastore from a single client's perspective [110]. Client-centric consistency models are generated (for example [120, 121]) based on the work on Bayou [122].

- *Monotonic Reads:* It is the first client-centric consistency model and a data store is monotonic-read consistent if satisfy the following condition:

  > "If a process reads the value of a data item x, any successive read operation on x by that process will always return that same value or a more recent value" [110].

  So, monotonic-read consistency guarantess that if a process retrieves a value of a data item A at time t_1, than the process never gets an older version of data item A at later time.

- *Monotonic Writes:* This client-centric consistency model ensures that write operations are performed in the correct order on all copies of the data store. If a data store is said to be monotonic-write consistent, it must satisfy the following condition:

  > "A write operation by a process on a data item x is completed before any successive write operation on x by the same process" [110].

- *Read Your Writes:* It is similar to the previous client-centric monotonic-read consistency model, and a data store is read-your-write consistent if the following condition is satisfied:

  > "The effect of a write operation by a process on data item x will always be seen by a successive read operation on x by the same process" [110].

  So a write operation is always completed before a successive read operation that performed by the same process independent from where the read operation is performed.

- *Writes Follow Reads:* Writes-follow-reads consistency model ensures that updates are disseminated to replicas as the result of previous read operations. A data store is writes-follow-reads consistent if the following condition is satisfied:

  > "A write operation by a process on a data item x following a previous read operation on x by the same process, is guaranteed to take place on the same or a more recent value of x that was read" [110].

2.3.3 Update Propagation and Consistency Protocols

An important design issue in update propagation is what to propagate to replicas. There are three possibilities to propagate:

- Propagate only notification to replicas.
- Transfer data from one replica to another.
- Propagate the update operation to other replicas.

A promising approach to our design would be propagating the data from one copy to another due to the nature of integrated annotation tools.

Another design issue is to decide whether updates are pulled or pushed. In a push-based protocol, updates are disseminated to all replicas without their asking for the updates. On the other hand, updates are retrieved from replicas by asking them at any moment in pull-based protocol. Furthermore, unicasting or multicasting communication approaches should also be decided to propagate updates. Because of the nature of the annotation tools, our proposed approach is to have a push and time-based pull approaches to propagate updates via unicast communication. Since annotation tools do not support publish/subscribe mechanism.

2.3.4 Related Projects

2.3.4.1 CVS

CVS (Concurrent Versions System) [123, 124] is a version control system that keeps the versions of files during their development period, allows users to edit a group of files collaboratively and retrieve old versions of them whenever requested. CVS allows several users to work on a shared file concurrently without loss of data. CVS has a

centralized communication mechanism via a single site and the repository that holds the main copies and previous versions of files is stored in the central site.

CVS uses optimistic replication approach for data sharing. Multiple users can have the copies of files and edit them concurrently. Each copy is private and local to each user. After the modifications are done to the files, users commit their copies to the repository. A commit is effective if nobody has changed the same files. If the same files have been modified and there is no conflict (overlapping modifications), then CVS merges those files automatically. Otherwise, users are notified that those commits need to be done manually and recommitted. Hence, CVS supports flexible collaboration and rare conflicts are resolved manually by the users.

CVS is a multimaster version control system since multiple users can perform modifications to files.

2.3.4.2 Wooki: Collaborative Editing System

Wooki [125] is a P2P wiki system based on the Woot [126] algorithm that ensures eventual consistency for linear structures in P2P environments. Main goal of the Wooki system is to replace the centralized architecture of a current Wiki system with a P2P network of wiki servers. In this approach, data stored on a wiki system is replicated over the P2P wiki servers and this raises up the question how to maintain consistency among the P2P wiki servers? Pessimistic replication approaches do not scale well compared to optimistic approaches. Furthermore, they also do not support offline collaboration whereas as pessimistic replication approaches do. Wooki uses optimistic replication approach to preserve consistency among replicas.

A wooki network consists of dynamic sites that can join or leave the network at any time. Each site has a unique id called "*siteid*" and site id values are totally ordered. Each site demands the partial knowledge of the whole network and keeps the replica of other sites in the network. A wooki system consists of three major components:

- Wooto: It is the core component and "it is in charge of generating and integrating operations affecting the documents" [125].
- Second component is responsible for the user interface.
- Third component "in charge of disseminating local operations and retrieving remote operations" [125].

A wooki system uses wooto algorithm to ensure eventual consistency among the replicated documents located at several sites. It uses lpbcast algorithm [127] to propagate operations to connected nodes and combines anti-entropy algorithm [128] to handle offline nodes.

The Wooki system has been implemented in Java as servlets in a Tomcat server and a wooki prototype is available from http://p2pwiki.loria.fr/.

2.4 Technologies

We have overviewed the main technologies that are crucial to our design and implementation model of our proposed thesis research in the following sub-sections.

2.4.1 Apache Axis 1.x

Axis is a Web Service container and available from http://ws.apache.org/axis/ . It is basically a SOAP (Simple Object Access Protocol) engine as defined in [7] including:

- A simple stand-alone server,

- A server which plugs into servlet engines such as Tomcat,
- Extensive support for the Web Service Description Language (WSDL),
- Emitter tooling that generates Java classes from WSDL.
- Some sample programs, and
- A tool for monitoring TCP/IP packets.

Key features of the Apache Axis 1.x defined in [7] are:

- *Speed*: Axis 1.x is faster than the earlier versions of Apache SOAP since it uses SAX (event-based) parsing.
- *Flexibility*: Architecture of Axis 1.x is flexible and provides developers with ability to add extenstions into the Axis engine for custom header processing, system management or any other custom needs.
- *Stability*: Axis is very stable since its defined published interfaces are change slowly when it is compared to other parts of Axis.
- *Component-oriented deployment*: Common patterns can be implemented to process custom applications in Axis 1.x by defining resusable networks of Handlers.
- *Transport framework*: Core of the Axis engine is transport-independent and Axis 1.x supports various protocols including SMTP, FTP, and message-oriented middleware.
- *WSDL support*: Axis 1.x supports the WSDL (Web Service Description Language, version 1.1). Users can easily build stubs to access remote services, and automatically export machine-readable descriptions of their deployed services from Axis.

Axis 1.x provides three main interfaces: (a) Remote Procedure Calls (RPC) [129]; (b) Document/wrapped; (c) Message style communications. In the RPC style, a Java object is serialized into XML and deserialized back into a Java object at the target point. It is very profitable to use the RPC style if a Java program has already been written and need to be deployed. Document and wrapped styles are similar to each other, whereas they are different due to their use of SOAP encoding. In document/wrapped style data is encapsulated within a plain XML document and serialization/deserialization operations are not required, but binding is necessary in this type of deployment. Finally, message style communication is a user-defined and it is very flexible due to its nature of being in an XML document and not necessity of serializers/deserializers in this type of style.

- **Discussion:** In our implementation of our thesis research, we have used Axis 1.2 version as our Web Service container. We have followed RPC style communication to provide access to our service interfaces as explained in 3.1. We have transferred messages between clients and services in XML format via RPC style communication. We preferred to use RPC style communication since we have already implemented our thesis research in Java language and we need to deploy our implementation as a Web Service, which is accessible from various clients running on various platforms.

2.4.2 Jakarta Commons HttpClient

HttpClient is an open source project that provides functionalities to access resources via HTTP protocol. It is available from http://hc.apache.org/httpclient-3.x/. Its main futures as described in detail in [130] are:

- Implementation of HTTP versions 1.0 and 1.1 in pure Java language.

- Implementation of the all HTTP methods (GET, POST, PUT, DELETE, HEAD, OPTIONS, and TRACE) in an extensible OO framework.
- Supporting encryption with HTTP over SSL (HTTPS protocol).
- Providing transparent connections via HTTP proxies.
- Providing tunneled HTTPS connections through HTTP proxies, via the CONNECT method.
- Allowing transparent connections through SOCKS proxies (version 4 & 5) using native Java socket support.
- Provides authentication through Basic, Digest and the encrypting NTLM (NT Lan Manager) methods.
- Ability to plug-in custom authentication methods.
- Ability to Multi-Part form POST for uploading large files.
- Pluggable secure sockets implementations, making it easier to use third party solutions
- Support for connection management in multi-threaded applications. It allows setting the maximum total connections and also the maximum connections per host. Furthermore, it can detect and close stale connections.
- Ability for Automatic Cookie handling. It also allows plug-in mechanism for custom cookie policies.
- It supports persistent connections by using KeepAlive in HTTP/1.0 and persistance in HTTP/1.1
- It allows direct access to headers and the response code, which are sent by the server.

- It allows setting connection timeouts.

➢ **Discussion:** In our design and implementation of thesis research, we need to have a mechanism to communicate explicitly with annotation tools to retrieve data from and send data to. Our CFDAR has been designed to work in the background as a multi-threaded form, and HttpClient is a perfect fit to provide our needs by supporting all HTTP methods, cookies, and authentication in a multi-threaded environment. Implementation details can be found in CHAPTER 5.

2.4.3 XML Parsers

There exist several parsers for XML processing. DOM and SAX parsers are the most popular ones, and DOM parser is the most widely used one for XML processing. It reads and validates the XML document. Document Object Model (DOM) [131] provides "a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents" [131]. Documents are represented in a tree structure in the DOM structure. Each node in the tree structure can be one of the specified types and what they may have as their children is specified in [131] as:

- *Document*: Element (maximum of one), ProcessingInstruction, Comment, DocumentType
- *DocumentFragment*: Element, ProcessingInstruction, Comment, Text, DATASection, EntityReference
- *DocumentType*: No children
- *EntityReference*: Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference

- *Element*: Element, Text, Comment, ProcessingInstruction, CDATASection, EntityReference
- *Attr*: Text, EntityReference
- *ProcessingInstruction*: No children
- *Comment*: No children
- *Text*: No children
- *CDATASection*: No children
- *Entity*: Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
- *Notation*: No children

Entire XML document represented in the DOM structure is kept in the memory and this allows developers to retrieve any element of the document easily. So, it is very profitable to use DOM parser in the case that a document needs to be accessed several times. On the other hand, it requires large amount of memory, and it gets worsen when an XML document gets bigger.

SAX (Simple API for XML) [132] parser does not work based on creating a document object tree like DOM parser does. SAX parser works as a stream parser with an event-driven API, where flow of the program is determined by events. Several methods can be defined by users to be called when SAX events occur during parsing a document. The SAX events include:

- XML Text nodes
- XML Element nodes
- XML Processing Instructions

- XML Comments

SAX parsers have some advantages and disadvantages when it is compared to DOM parser. SAX parsers use much smaller memory than DOM parsers since DOM parsers keep the whole document as DOM structure in the memory. So, size of the memory that needs to be used by DOM parsers is depending on the document, whereas in SAX parsers the amount of memory is based on the maximum depth of the XML tree. It is always smaller than the parsed document as a tree itself. Hence, SAX parsers work faster than DOM parsers but the document needs to be parsed again and again to re-read the parsed data. So, it does not provide an efficient way to parse XML documents when a document needs to be accessed many times.

2.5 Summary

This chapter discussed the background information, reviewed the related work and surveyed the major technologies related to our research. First, an overview of Web 2.0 and Annotation Tools, and its related projects are given. Second, event systems and the related projects are reviewed. Third, consistency maintenance approaches for distributed systems are summarized and discussions are given throughout the chapter. Finally, the related technologies are presented. From this, we have identified useful strategies that we will use in our research architecture.

CHAPTER 3 Event-based Infrastructure

CHAPTER 2 surveyed event systems, and analyzed the existing tools and approaches that provide web-based services to store, share, and tag various resources among small groups, teams and communities and their limitations involved in representing scholarly publications and communicating with each other. Based on the analysis, this chapter particularly focuses on the modular architecture of a system by addressing the first part of the research problems given in Section 1.2.

3.1 Design Overview

In this chapter we introduce a novel architecture that is designed to provide an ideal approach to unify and federate major annotation tools, support collaboration, represent and manage content of scientific documents coming from various sources in a flexible fashion. General architectural design for the proposed Event-based Infrastructure

(EBI) and Consistency Framework for Distributed Annotation Records (CFDAR) appears in Figure 3-1.

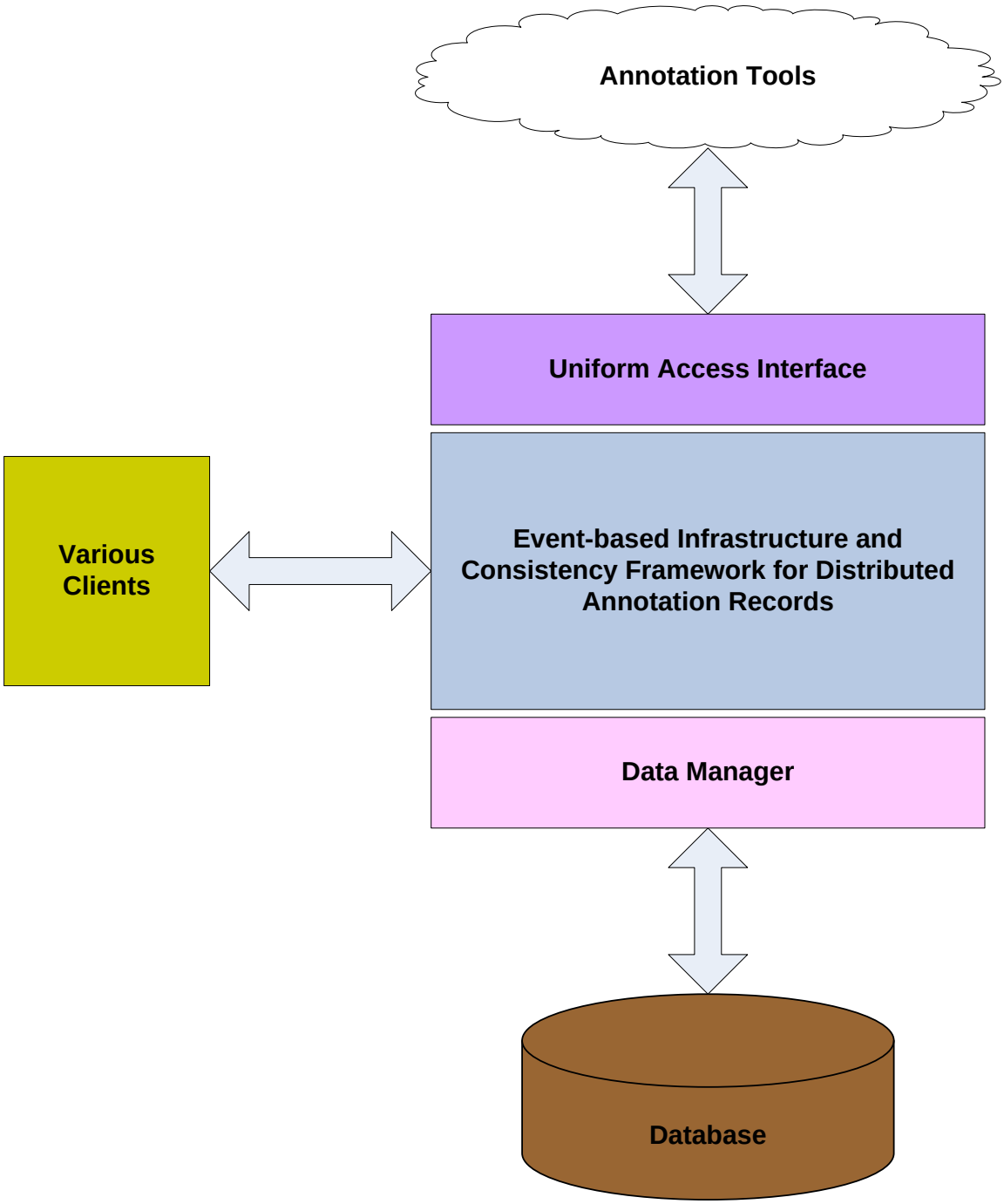

Figure 3-1: General Architectural Design for the Event-based Infrastructure and Consistency Framework for Distributed Annotation Records

To meet the requirements for handling data and metadata coming from different sources such as online collaboration tools, peer to peer systems, social bookmarking websites, academic search engines, scientific databases, journal and conference content management systems, the Event-based Infrastructure utilizes the use of event concept as its building blocks. According to this concept, content of scientific documents originating from various sources are represented as events. Events constitutes the base unit for our Event-based Infrastructure, and an *event* is commonly defined as the act of changing the value of an attribute of some object [133]. Storing all the events about an object enables the actions on this object to be reviewed and undone [134]. An event may also be defined as an action with a time stamp and a message [106]. In our Event-based Infrastructure, we adopt the view of an event as a time-stamped action on a document, which only maintains the modifications to an object. We distinguish between *minor* and *major* events: insertion of a new digital entity (DE), which is a collection of metadata representing a scholarly publication, into the system or deletion of an existing digital entity from the system is considered a major event; modifications to existing digital entities are considered minor events. Examples of modification are: deleting one or more fields of a digital entity, changing the value of one or more fields of a digital entity by adding or deleting metadata, and so on. Another concept underlying the Event-based Infrastructure is that of *dataset*. A dataset is a collection of minor events related to a user. A dataset creation is a way to group the modifications of a digital entity. There are two important issues requiring attention during the process of dataset creation (described in Section 3.7.3.1.1 in detail): (a) Events that are selected as members of a dataset must belong to the same digital entity (we do not want to include into a dataset events

belonging to different digital entities). (b) The order of the events is a key factor in that the events related to a DE are applied in the order they occur. A document representation by events is depicted in Figure 3-2. As it is seen on the figure, documents are constructed from major and minor events. A major event represents the original entry in the system, while the minor events are the modifications to the original entry during the time.

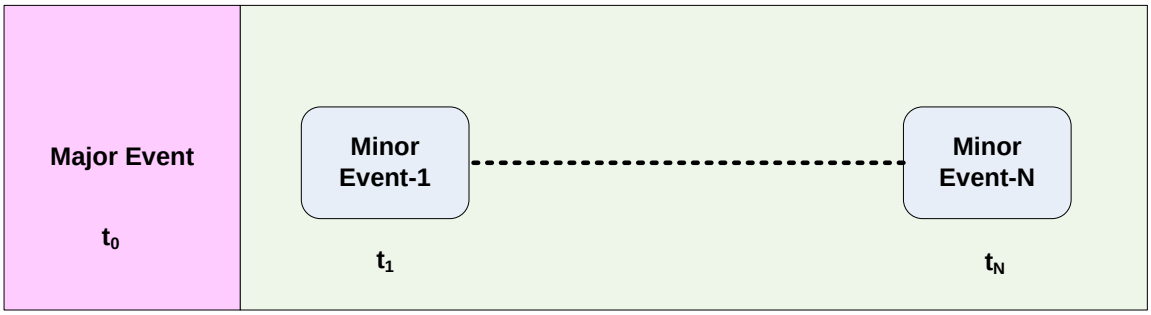

Figure 3-2: Document Representation in Event-based Infrastructure

To meet the uniformity requirements, Event-based Infrastructure enables the proposed system to support one to many annotation tools interactions and their communication protocols. This way, the system unifies different annotation tools under one hybrid system.

To meet the federation requirements, it presents a federation capability where different annotation tools and their services can be federated in metadata instances. To enable this capability, we introduce annotation tools schema by integrating different annotation tools data models. The annotation tools schema provides a common platform to enable interaction between the annotation tools.

Table 3-1: Stored Metadata Comparison in Annotation Tools

Stored Metadata	Citeulike	Connotea	Delicious
URL	✓	R	R
TITLE	R	✓	
DOI	✓	✓	
PMID		✓	
ISBN/ASIN		✓	
REFERENCE TYPE	R	✓	
AUTHORS	✓	✓	
PUBLICATION NAME		✓	
VOLUME NO	✓	✓	
ISSUE NO	✓	✓	
CHAPTER	✓		
EDITION	✓		
START PAGE	✓		
END PAGE	✓		
PAGES		✓	
YEAR	✓		
MONTH	✓		
DAY	✓		
PUBLICATION DATE		✓	
DATE OTHER	✓		
EDITORS	✓		
JOURNAL	✓		
BOOK TITLE	✓		
HOW PUBLISHED	✓		
INSTITUTION	✓		
ORGANISATION	✓		
PUBLISHER	✓		
ADDRESS	✓		
SCHOOL	✓		
SERIES	✓		
BIBTEX KEY	✓		
ABSTRACT	✓		
DISPLAY TITLE		✓	
TAGS	†	R	✓
TAG SUGGESTIONS		✓	
DESCRIPTION		✓	R
MY WORK		✓	
EVERYONE'S TAG	✓		
PRIVACY SETTINGS	✓	✓	
RELEASE DATE TO ALL USERS		✓	
PRIORITY OF RECORDS	✓		
NOTE	✓		✓
COMMENT		✓	

✓ = Supported, R = REQUIRED, † = Adds "no-tag"

To meet the comprehensive metadata field requirements, Event-based Infrastructure support various metadata fields to represent the complete metadata about a scholarly publication. Supported metadata fields are compatible with the one that specified by the Dublin Core Metadata Initiative (DCMI) [135] and BibTex [41]. Table 3-1 portrays the stored metadata comparison in Connotea, Citeulike, and Delicious annotation tools.

To meet the requirements for providing a flexible architecture to track modifications to documents and metadata that collectively form a digital entity, Event-based Infrastructure benefits from representing modifications to documents as events. Events are time-stamped entities that encapsulate the changes to documents. Associated with each digital entity, there is an initial set of metadata. This initial set of metadata is represented by a major event, and it may come from different sources. DE metadata of a record at a certain point is the result of applying all the available ordered datasets to the initial digital entity metadata. Another word, by replaying events, it is possible to reconstruct a DE at any point in its evolution (explained in detail in Section 3.7.3.1.2).

To meet the flexible choices for version control, Event-based Infrastructure provides an architectural component called Rollback Module. The rollback module was designed to retrieve the desired version of a DE by undoing the necessary events that are already applied to a DE. The rollback mechanism is explained in Section 3.7.3.4.

To meet the requirements for providing services to extract data from annotation tools to a specified repository and to upload data from a desired repository to an annotation tool, Event-based Infrastructure uses Communication Manager that implement

the uniform access interface abstraction layer explained in Section 4.4.2.2 to communicate with annotation tools to exchange data and metadata.

To provide convenient structure for organizing data and metadata without dealing with the complexity of hierarchical structure, grouping and accessing related documents easily, Event-based Infrastructure support collaborative tagging of documents.

To meet the interoperability requirements, Event-based Infrastructure has been designed as a service-enabled system. Web Services enables the interoperability between different software applications running on different platforms [136]. Web Services have an interface which is described in a machine-processable format, and Web Services support interoperable machine to machine interaction over a network. Web Services are defined in a language called Web Services Description Language (WSDL) [137]. The clients can communicate with a web service by exchanging messages in SOAP (Simple Object Access Protocol) format. SOAP [138] is a platform and language independent communication protocol for exchanging information in distributed environment. SOAP is an XML based protocol, and consists of three parts the envelope, the encoding rules, and the Remote Procedure Call (RPC) convention. SOAP can be used in any combination of with some other protocols such as HTTP [139], FTP [140] etc. WSDL is specified in XML, and it is used for describing and locating Web Services. WSDL uses four major elements to define Web Services:

- portType: Specfies the operations performed by the web service.
- message: Message defines the data elements of an operation.
- types: Types element defines the data types used by the web service.

- binding: Binding specifies concrete protocol and data format specifications for the operations and messages defined by a particular portType.

As a summary, our Event-based Infrastructure supports: (1) Representation of data and metadata coming from various sources as events; (2) Tracking changes to data and metadata; (3) Supporting the extraction and collection of metadata and documents into a central repository from various sources such as online collaboration tools, peer to peer system, social bookmarking websites, academic search engines (Google Scholar (GS) [141] and Windows Live Academic (WLA) [142]), scientific databases, journal and conference content management systems. So, it is inevitable to have an Event-based Infrastructure to support and represent those multiple sources of metadata information for digital entities in a collaborative environment; (4) Supporting unification and federation of major annotation tools. It also provides services to upload data and metadata from a specified repository to annotation tools; (5) Supporting sharing, modifying, and collaboratively tagging of DEs; (6) Supporting the communication of annotation tools with each other via Web Service technology; (7) Providing comprehensive metadata support to be able to represent the whole metadata of a scholarly publication; (8) Optimistic replication approach to ensure eventual consistency between replicas as explained in detail in CHAPTER 4; and (9) Supporting flexibility to maintaining all versions of DEs and ability to rollback to any version by undoing the related events.

3.2 Content of a Digital Entity (DE)

In our proposed arhitecture, a DE is divided into its metadata that represents a scholarly publication. Another word is that a DE consists of several metadata fields that hold the related metadata of a scholarly publication. In our Event-based Infrastructure,

each DE's metadata is designed to be stored in a relational database (MySQL) and the latest version of a complete DE can be generated from scratch by executing all of its minor events and its major event in the order they are created. Figure 3-3 represents a DE and its encapsulated metadata that represents the complete metadata of a scholarly publication in our proposed architecture.

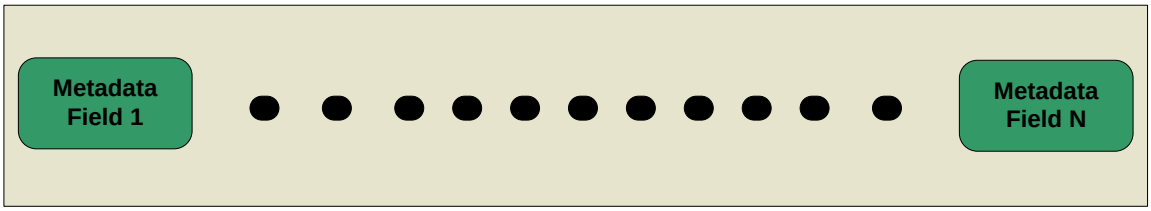

Figure 3-3: Content of a Digital Entity

3.3 Storage of a Document as a DE

In our Event-based Infrastructure, initial metadata of a scientific document is represented by a major event, while further updates to any metadata field of an existing DE are represented by minor events. Both major and minor events, which forms DE and contains the metadata fields of the document, are stored into a MySQL system database with a unique DE id and its owner information.

Event-based Infrastructure requires associating each DE with a folder, which is a label representing a logical repository to collect DEs into. Each DE has an owner, and an owner is a registered user with the system. Hence, each DE is tied to a user, an owner of a DE can define access rights for his/her DEs for other users, and groups as read, write, and execute. Moreover, an owner of a folder can also specify access rights for his/her folders for other users, and groups as read, write, and execute. Depend on the defined access

rights for DEs and their belonging folders, our Event-based Infrastructure support users or communities to collaborate on the specified DEs.

3.4 Duplicate Detection

It is a critical issue to find out if a document that is about to be inserted into the system already exists in the system or not. Our Event-based Infrastructure defines its own duplicate detection algorithm to decide whether two given DE is similar or not with a defined threshold value.

Our Event-based Infrastructure's duplicate detection algorithm has been designed to compare a given digital entity with all the DEs from a repository to find its matching primary copy whether it exists in the database or not. Our duplicate detection algorithm works based on hashing (by MD5) the available metadata fields of a given document including URL, title, authors and publication venue.

3.5 Event-based Infrastructure Update Model

Our proposed Event-based Infrastructure's update model is built on the event-based structure to provide flexible choices to users. Our update model uses events for applying the updates on existing digital entities. It provides users with flexible choices to apply the updates as minor events when faced with existing DEs within the repository as:

- Keep the existing version.
- Replace the existing version with the new one.
- Merge the existing and the new version.

Our update model supports the above update model concepts to be applied for all matching digital entities or each existing individual digital entity in the system. By doing

that, updates can be applied to each individual or all digital entities as a default based on the selected choice.

3.6 Supported Annotation Tools

The Event-based Infrastructure presents an architecture, which extends the capabilities of the existing major annotation tools (Connotea, Citeulike, and Delicious). This capability enables unification and federation of annotation tools in our research framework.

To utilize the unification capability, we provided implementations of the three annotation tools: Connotea, Citeulike and Delicious. Thus, the prototype implementation of the Event-based Infrastructure and Consistency Framework for Distributed Annotation Records provides a uniform access interface described in Section 3.7.1 and supports the Connotea, Citeulike and Delicious services with added capabilities.

To utilize the federation capability, we introduced annotation tools schema specification. With the annotation tools schema, we integrate Connotea, Citeulike, and Delicious service data models under one unified schema. We have defined an XML Schema to combine metadata models of the Connotea, Citeulike and Delicious annotation tools.

3.6.1 Annotation Tools Schema Semantics

Annotation tools schema represents the merged schemas from the major annotation tools by integrating their schemas into one. Schema integration is a functionality of providing a unified representation of multiple data models [143]. In our annotation tools schema specification, we have merged data model schemas from the

major annotation tools. The Event-based Infrastructure and Consistency Framework for Distributed Annotation Records prototype specifies a merged schema for Connotea, Citeulike and Delicious annotation tools.

3.7 Overview of the Architecture Components

The detailed architectural design of the Event-based Infrastructure and Consistency Framework for Distributed Annotation Records is depicted in Figure 3-4. Event-based Infrastructure's modules can be categorized under five main categories: a) Uniform access interface; b) Event-based Infrastructure services; c) Digital Entity Manager; d) Timestamp Generator; e) Data Manager. Annotation Tools component is explained in Section 4.4.2.1, Annotation Tools Update Manager is explored in Section 4.4.2.3, and Communication Manager is discussed in Section 4.4.2.2.

3.7.1 Uniform Access Interface

The uniform access interface presents a common access interface to the Connotea, Citeulike, and Delicious annotation tools. Another saying is that the uniform access interface imports API of the supported annotation tools. The Event-based Infrastructure and Consistency Framework for Distributed Annotation Records prototype supports API for Connotea, Citeulike and Delicious. The access interface can import more APIs, as the new annotation tools are integrated with the system. These major annotation tools have been unified under one unified-architecture called Event-based Infrastructure and can be accessed through the uniform access interface. The uniform access interface layer is implemented and managed by the Communication Manager as explained in detail in Section 4.4.2.2.

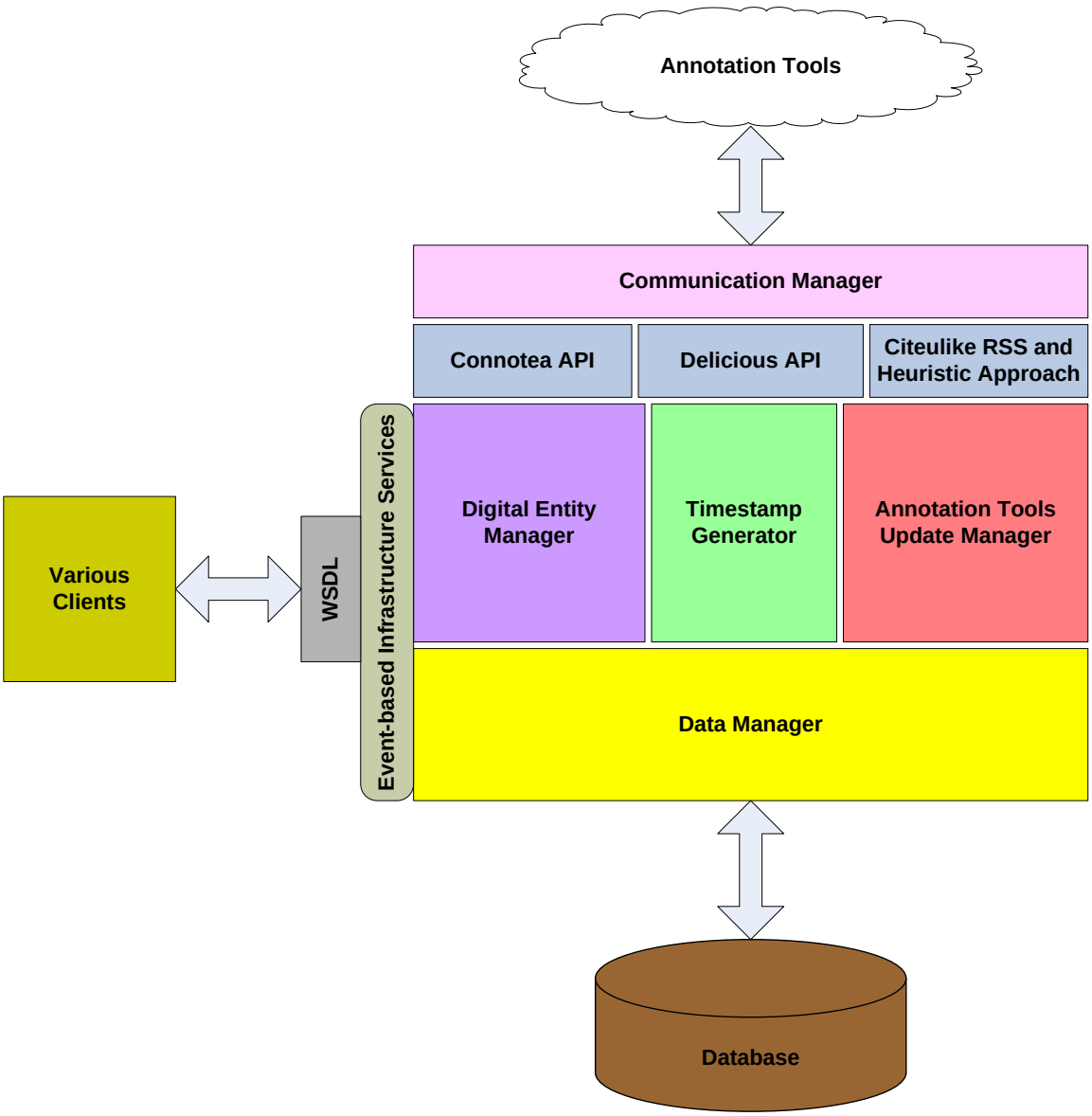

Figure 3-4: The Architectural Design for the Event-based Infrastructure and Consistency Framework for Distributed Annotation Records

3.7.2 Event-based Infrastructure Services

Event-based Infrastructure services are the Web Services that provide a communication with the core services of the proposed architecture over HTTP protocol through SOAP messages. It has an interface written in WSDL and it is used for describing and locating the Event-based Infrastructure service implementations. The

architecture supports seamless access to archival data and metadata through standard Web Services interfaces.

Supporting Web Service technology provides many advantages. “A Web Service is an interface that describes a collection of operations that are network accessible through standardized XML messaging” [144]. The interface hides the implementation logic and details from the users, and this allows the service to be used on different platforms rather than the one it was implemented. Also any application capable of communicating through the standard XML messaging protocol and regardless of with which programming language it was implemented in can use the service through the standard interface. These properties allow Web Services based frameworks to be loosely coupled and be component oriented. Due to the standard interfaces and messaging protocols the Web Services can easily be assembled to solve more complex problems. One significant feature of the Web Services is that they allow program-to-program communications. The main difference between the Web services and the other component technologies is that, the Web services are accessed via the ubiquitous Web protocols such as Hypertext Transfer Protocol (HTTP) and Extensible Markup Language (XML) instead of object-model-specific protocols such as Distributed Component Object Model (DCOM) [145] or Remote Method Invocation (RMI) [53].

Finally, the capabilities that provided by Web Service technology can be of great benefit to the usage of our proposed Event-based Infrastructure services. Many applications running on different platforms can access its services via its service interfaces.

3.7.3 Digital Entity Manager

Digital Entity Manager is an umbrella name for a group of modules that contributes to DE management together. Its modules are: (1) Events and Dataset management; (2) Digital Entity Update Management; (3) Periodic Updates Management; (4) History and Rollback Management. Figure 3-5 displays the Digital Entity Manager and its components. The details of each module are given in the following sections respectively.

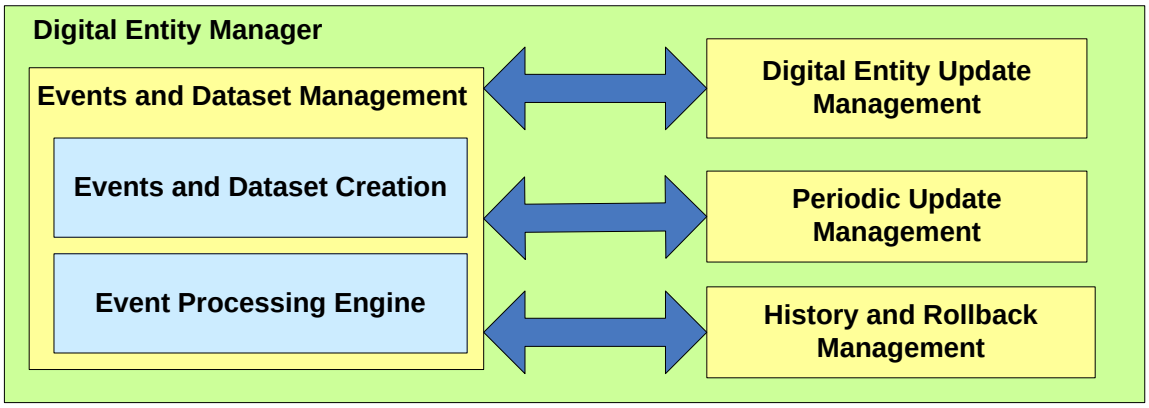

Figure 3-5: Digital Entity Manager

3.7.3.1 Events and Dataset Management

Events and Dataset Management module is responsible for the creation of major/minor events and datasets. It has two submodules to provide its main services:

3.7.3.1.1 Events and Dataset Creation

In order to create a new DE or to validate modifications to a DE, major or minor events need to be created by the proposed system. Insertion of a new document and its metadata are represented with a major event stored into a MySQL database with a unique id in the proposed architecture. When a modification made to any metadata field of an existing DE, then only the modification is saved into the repository (MySQL database) as

a minor event with a unique event id and the unique DE id that this event belongs to. Desired minor events belonging to the same DE can be grouped together to form a dataset with a unique dataset id assigned by the system automatically. Created events and datasets are processed by Event Processing Engine to build a desired version of a DE at any given time. Details of the event and dataset processing can be found in Section 3.7.3.1.2.

There are many key advantages of using events and datasets for representing data and metadata. First, by representing data and metadata with events and datasets, our proposed system never loose a copy of a DE for any version. Since, data and metadata are archived as events into a MySQL database, and any version of a documents can be rebuild from scratch by using its major event and datasets that brings collection of minor events. Another word, using event structure for representing data and metadata enables our proposed architecture to rollback to any version of a document in its history. Second, timestamp information is a crucial thing for distributed and collaboration systems. Each event has a timestamp value and this information can be used for ordering the events or maintaining consistency among documents by defining the order of update operations on primary copy and replicas of data and metadata. Third, events provide our architecture to be flexible for building and retrieving a document based on the desired user's or group's events. Finally, due to the nature of our documents represented by several metadata fields, using an event structure leverages collaboration and updates on the metadata fields of a document. Our architecture can only store the modified metadata field of a document instead of storing the whole modified document, and the complete document can later be constructed by using its events based on their occurrences in time.

3.7.3.1.2 Event Processing Engine

Main duty of the Event Processing Engine is to build a complete document by using the document's dataset and events for a given state. To do so, Event Processing Engine collects all the dataset and the events belong to the requested DE from a MySQL database. Having done that, Event Processing Engine process all the minor events sorted by time using their timestamp on top of the major event to retrieve the final version of the requested document. Another word, by using the initial metadata, which is a major event, of a digital entity and by applying dataset(s) on top of it, one can retrieve any version of a DE. Hence, in case of an error or users' request, our architecture supports to restore the system to a previous safe state by using the related dataset for that state.

The example in Figure 3-6 shows the process of building a document by using its major event and datasets. Each dataset (Dataset-1… Dataset-N) is composed of a number of minor events, and each dataset modifies the digital entity metadata based on the events that it has. In our proposed Event-based Infrastructure, all available datasets of a digital entity are applied on top of the initial digital entity metadata, which is the major event of this DE, based on their increasing creation time to retrieve the latest digital entity metadata. During the application process, we apply each dataset and its associated events in the increasing order of their creation time.

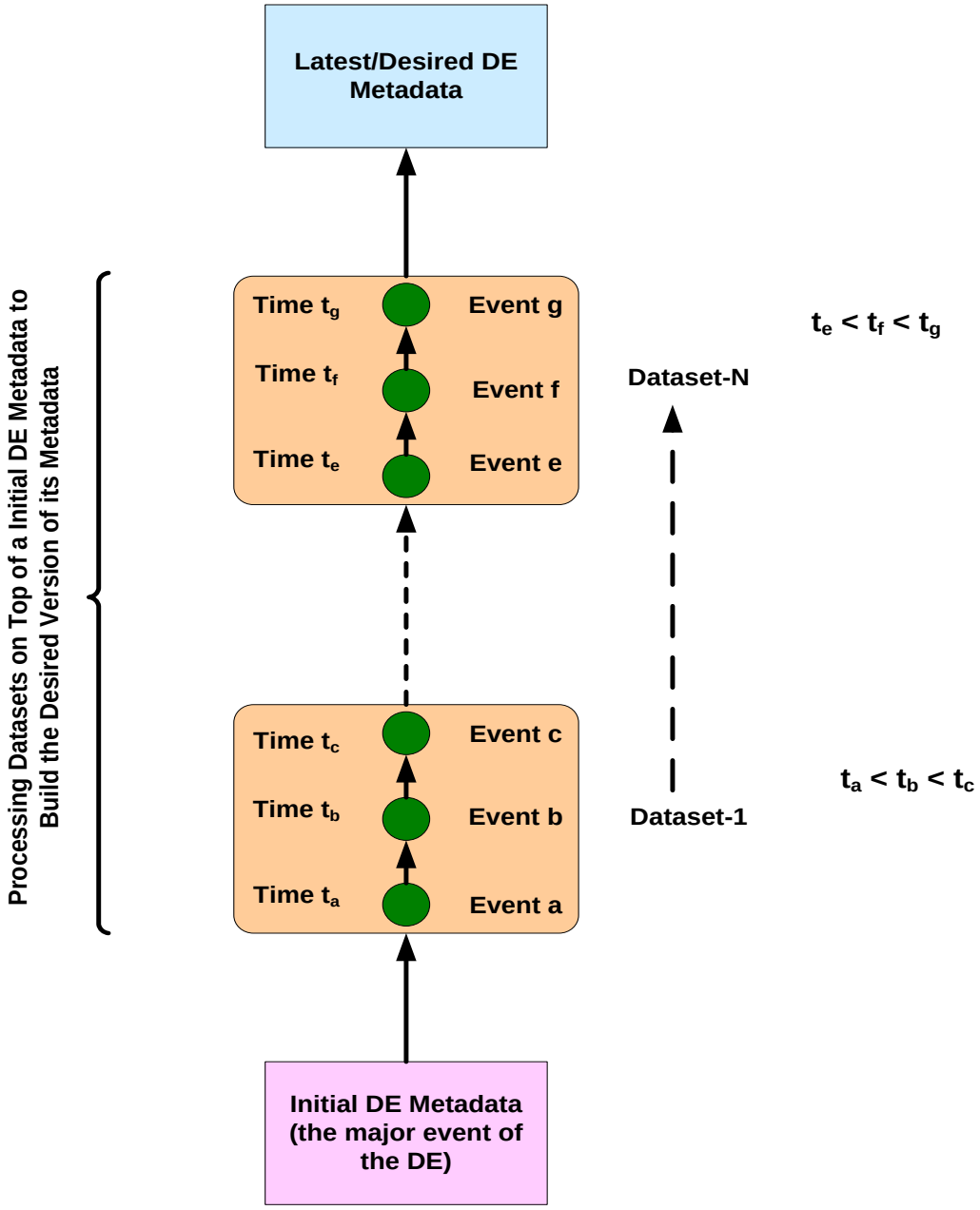

Figure 3-6: Retrieving the latest digital entity metadata

As depicted in Figure 3-6, to build a digital entity metadata for a certain point, we just apply the related dataset(s) on top of the initial digital entity metadata based on their creation time, and the plus sign (+) in the formula indicates the application of the related dataset(s) on top of the initial digital entity metadata. As a result, we have:

$$\text{Current DE Metadata} = \text{Initial DE Metadata} + \sum_{k=1}^{n} \text{Dataset (k)}.$$

3.7.3.2 Digital Entity Update Management

Our proposed Event-based Infrastructure stores data and metadata representing metadata of scholarly publications into a MySQL database in the form of events, which are the primary copies of the DARs stored at the annotation tools, and it supports data exchange between the MySQL database and annotation tools. An update is specified as either a major or a minor event in the proposed system based on what it has modified within the system. In a major event representation, an update creates a new entry or deletes an existing entry, while in a minor event representation an updated modifies any metadata field of a DE [3]. Digital Entity Update Management module deals with the updates that are made directly on the primary copy of each DE via Event-based Infrastructure services. Its main responsibilities are: (1) Deciding the event type (major or minor event) by using duplicate detection mechanism to find out whether this update is trying to create a new DE or update an existing DE; (2) Creating an update event that could be a major or minor event by setting up necessary parameters into a MySQL database via Events and Dataset Creation module explained in Section 3.7.3.1.1; (3) Passing the update event and its data to Communication Manager to be propagated to the supported annotation tools to either upload a new entry in the case of major event or reflect the changes on replica copies in the case of a minor event via Communication Manager as explained in Section 4.4.2.2. Digital Entity Update Management module provides a push based approach to propagate an update immediately to the integrated annotation tools when it occurs. Updates are disseminated to the integrated annotation

tools via unicast communication strategy; (4) Returning a confirmation result to the clients in XML format.

Each minor event is defined with its parameters including its unique id, its operation type (replace, merge, delete), which DE it belongs to, its timestamp value, and its data. These parameters are transferred as XML message to the necessary modules. Schema of parameters of an update event is depicted in Figure 3-7.

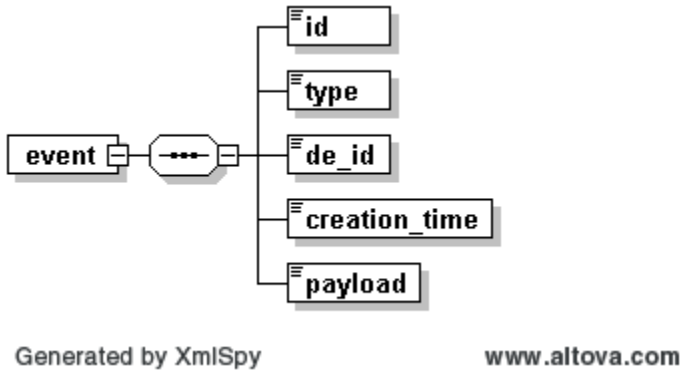

Figure 3-7: Update Event Parameters

Finally, Digital Entity Update Management service can be accessed by any client that capable of communicating through XML messaging via Web Service call using SOAP messages over HTTP protocol.

3.7.3.3 Periodic Update Management

Periodic Update Management module keeps tracking the updates (minor events) to DEs, and it is responsible for retrieving and applying all the updates made in the system to same or similar DEs.

It has been designed to provide conveinent choices to retrieve the updates made to DEs belong to other users in order to execute them on users' own DEs based on our

proposed Event-based Infrastructure's update model as defined in Section 3.5. It detects the changes for updated digital entities as events and provides all available updates from other users for a user's all existing DEs. By keep tracking the updated events, Event-based Infrastructure can provide the owner of the digital entities with ability to update his/her digital entities with the new updated ones. Hence, users can collect and apply the updates, which were made on DEs by other users, for their own DEs by the provided Periodic Update Management service of the proposed system.

3.7.3.4 History and Rollback Management

Collaborative systems allow people to work together on a common task and share resources to pursue their goals. A mechanism to avoid undesired changes in the system is a critical issue in such systems. Because people work on a common set of resources, they could modify the same resources. So, data is exposed to unintentional user mistakes.

To avoid undesired changes and to have flexible choices in the system, it is necessary to have a mechanism for restoring the system to any previous state. There are several existing systems that provide mechanisms for restoring the state of the system to any previous state. For example, in the Windows XP operating system, if the system crashes, then the tool called "System Restore" can be used for restoring the system to the last working point. As another example, many developers of the same project works on the same source code and they use one of the versioning systems such as *Concurrent Versions System* (CVS) [146] or *Subversion* (SVN) [147] to access and submit their changes. They do modifications on the code and they submit their changes into the repository. If any of the developer needs to retrieve the previous version of the code, then they can obtain it through the versioning system that they are using in their project. As a

final example, Wiki systems allow their users to add, remove, change and edit a common digital content. By using “Recent Changes” page and “Revision History” function from the change log are being used for restoring the previous version of the content [12].

To allow the state of the system to be restored to any previous state, proposed Event-based Infrastructure system supports a service that lists the history of each DE and provide a mechanism to undo any changes (rollback) to the desired state in its history.

3.7.4 Timestamp Generator

Timestamp generator provides a service to generate unique timestamp values to the requesting processes. These unique timestamps values are used for ordering processes to execute them. Furthermore, events also need to be timestamped in order to impose an order on them.

3.7.5 Data Manager

Data manager is responsible for executing read or write requests on a data item. Data manager is not concerned about what operations it is performing. It just executes the coming operation on a data item.

3.8 Summary

In this chapter, we explained the proposed Event-based Infrastructure, its components and the representation of metadata of scientific documents as events. Our Event-based Infrastructure has a moduler architecture which improves the maintanence and simplicity of the system. The modules can be classified into five sub-groups. The first group is Uniform Access Interface that presents a common access interface to the Connotea, Citeulike, and Delicious annotation tools. The second group that contains the

service module which provides an interface to communicate with Event-based Infrastructure services over HTTP via SOAP calls. The third group is in charge of the Digital Entity Management. This module consists of 4 sub-groups. The first group is Event and Dataset Management and it is responsible with the creation and management of the events and the datasets. The second group is Digital Entity Update management and it deals with the updates made on primary copies of a DE. The third group is Periodic Update Management module and it provides a mechanism to collect and apply the updates made to a DE belongs to another user. Finally, History and Rollback Management part maintains the histories of DEs and allows users to rollback to any state of a DE in its history. The fourth group is the Timestamp generator that generates unique timestamp values for the requesting processes. The fifth group is the Data Manager, and it is responsible for executing the coming requests on data items.

CHAPTER 4

Consistency Framework for Distributed Annotation Records

CHAPTER 2 analyzed the major consistency maintanence approaches for distributed systems in detail, and CHAPTER 3 explained the first part of the proposed approach to represent, manage and deal with resources coming from various sources for scientific research. Based on the analysis, this chapter particularly focuses on modular architecture of a system by addressing the rest of the research problems given in Section 1.2. In the remainder of this chapter, we explain our Consistency Framework for Distributed Annotation Records (CFDAR) that is based on optimistic replication approach [124, 148] to ensure eventual consistency between replicas and provide the detail explanation about its modules.

4.1 Design Overview

We have designed a novel consistency framework that adopts optimistic replication approach to ensure eventual consistency between annotation tools where replicated Distributed Annotation Records (DARs) are stored. Our proposed framework CFDAR supports collaboration among DARs, which are replicas of the same document, kept at various web-based annotation tools. An overview of the proposed architecture design appears in Figure 3-4.

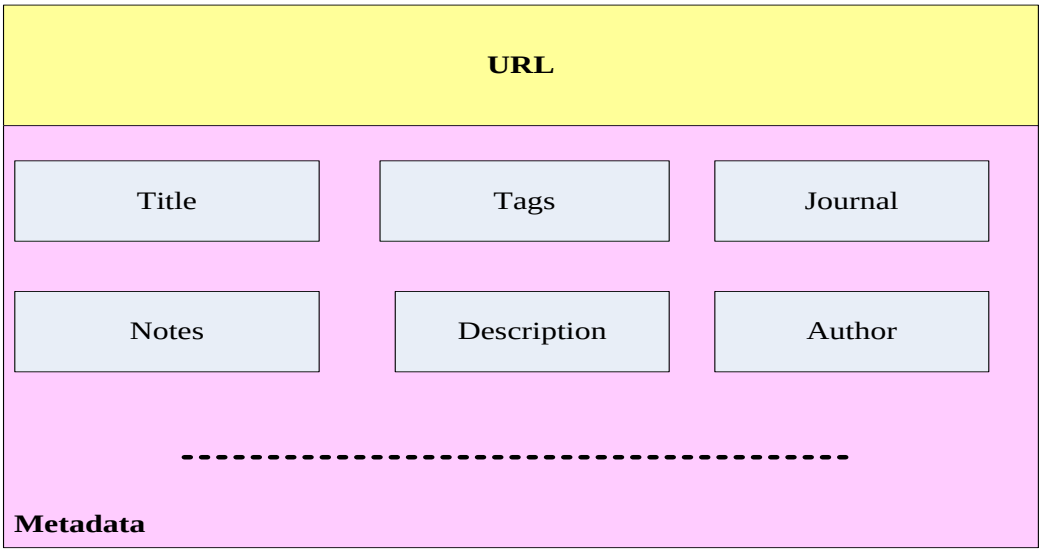

Figure 4-1: General View of a Distributed Annotation Record (DAR)

Annotation tools are one of the major Web-based Web 2.0 applications that provide a new way of sharing between users and communities. Users of those annotation tools can have the same documents in their account in several annotation tools. Those tools provide their users with ability to: (1) enter a new record; (2) delete an existing record; (3) modify an existing record; (4) tag their record; (5) share the content of their records with other users. URL value of each DAR is a mandatory attribute in these tools

and it is used as a unique key element for storing records in annotation tools. Figure 4-1 depicts general picture of a DAR that is held at annotation tools.

One major problem with annotation tools is that they do not provide timestamp information for the updated records, and eventually modifications to a DAR can come from multiple sources. The consistency concept arises when records get updated with unknown timestamp, and multiple copies (replicas) of a same document can be existed in different versions within the various numbers of annotation tools. Providing consistency maintenance is a fundamental issue [149], and our thesis research focuses on how to design a consistency framework to maintain consistency between the replicated DARs held on those annotation tools and their primary copies with additional information attached to them. The design of such an environment should consist of the group of annotation tools intended to be consistent with each other, and a main system, where a primary copy of each document from each annotation tools are stored with additional metadata information into a relational database (MySQL database).

Another fundamental issue with them is that annotation tools are lack of services or mechanisms to provide their clients with notification services for deleted, modified or new entered entries into their system. Because of this, there is no way to be notified about any changes in those systems. The only way to identify any changes in those tools is to have a external mechanism to go and check them periodically (Pull-based approach).

We have designed the CFDAR, which benefits from having an event-based infrastructure as its base, and timestamped events can thus be used to reconcile the system state, to be able to: (1) Run for consistency enforcement in the background; (2) Communicate with the integrated annotation tools periodically; (3) Retrieve records from

the annotation tools via pull based approach; (4) Compare records with their primary copies coming from the central MySQL database. (5) Collect updates based on the comparisons and put the updates for existing DEs in the system database into an update event list as minor events. If a record does not have a primary copy, then put it into the update event list as a major event. (6) Pass the found events to Digital Entity Update Management module to be inserted into MySQL database as events and disseminated to the integrated annotation tools via push based approach as explain in 3.7.3.2. Furthermore, users can collaborate on the primary copy of each DAR with each other by sharing the same document. And our consistency framework propagates updates made on a primary copy of a DAR immediately to each annotation tool to reflect the changes on all replicas via Digital Entity Update Management module explained in 3.7.3.2.

As a summary, our proposed CFDAR has been designed to adopt optimistic replication approach, and in the background it utilizes pull and push-based mechanisms to collect and propagate the updates to the integrated annotation tools so that the eventual consistency among the integrated annotation tools can be preserved. We are going to explain our CFDAR and its components in detail in the following sections of this chapter.

4.2 Consistency Criteria

The consistency maintanence issue has to do with ensuring that all copies of the same data to be the same at a given time. Some approaches to maintain consistency are discussed in detail in [110, 149-153]. Tanenbaum [110] differentiates consistency under two main categories: (1) data-centric; and (2) client-centric. In data-centric approach, all copies of data are updated whether some clients is aware of those updates or not. In client-centric approach, consistency is maintained from a client's perspective. Client-

centric consistency model allows copies of data to be inconsistent with each other as long as the consistency is ensured from a single client's point of view.

The implementation of the consistency models can be categorized as primary-based protocols (primary-copy approach) and replicated-write protocols [110]. In primary copy approach, updates are executed on a single location, and propagated replicas from there, while in the replicated-write approach; updates can be originated from multiple locations. For an example, techniques for maintaining consistency in P2P networks: (1) Push: Owner-initiated Consistency: messages are propagated through the P2P overlay in push approach; (2) Pull: Peer-initiated Consistency mechanism, individual peers polls the owner to figure out if a file is stale or not; and (3) Hybrid Consistency mechanism. Our approach enhances the popular consistency techniques, which had been originally designed for the distributed replicated systems, to be applied to DARs to maintain consistency among web-based annotation tools. Detailed background information regarding to consistency maintenance can be found in Section 2.3.

4.3 Exceptions in Concurrent Updates

Integrated annotation tools are independent external tools. When there is a concurrent update on both these tools and our proposed system where primary copies are stored, one of the update has to be lost. The only way to prevent loosing an update is having these integrated tools to have notification system such as publish/subscribe mechanism. In our domain it has very low possibility to have concurrent updates on a same record, and our architectural design utilizes optimistic replication approach to ensure eventual consistency. Our design relies on the following assumptions:

- To provide consistency for the concurrent updates, integrated annotation tools are required to support notification systems. Otherwise, an update made on a primary copy through the proposed system is considered as a valid update.

- Our proposed system check updates on annotation tools periodically by utilizing a pull approach. We assume that there can be maximum one modification to a record between each periodic consistency maintanence managed by the proposed consistency framework. If there are more than one updates, then our proposed system retrieves the last one as the one during the next periodic consistency maintenance check. Hence, our proposed system supports the updates occurred after our proposed consistency framework' update checking frequency. Annotation tools that require lower update frequency need to support a notification system. Furthermore, an update operation that is requested on a primary copy initiates an update check before starting the update execution between the primary copy and its replicated copies located at the integrated annotation tools to synchronize them.

- We assume that timestamp information for each updated record is the time that our proposed consistency framework checks annotation tools for retrieving updates periodically as specified in the configuration file.

- We assume that if there are updates coming from different annotation tool for a same record, these updates are count as concurrent updates. And, they are subject to our concurrent update policy. Hence, if the updates do not conflict, then they are applied on the record. If they conflict, then they are logged and request users to manually resolve them. Please see Section 4.4.1 for defined conflicts in our proposed system.

4.4 Consistency Framework

Our CFDAR has been designed to ensure eventual consistency between DARs kept at annotation tools and a primary copy of each DAR located in a central MySQL database. The consistency framework is a client-centric consistency model, and the implementation protocol is the replicated-write protocol since updates can be originated from several replicas. We have adapted the optimistic replication approach to ensure eventual consistency between the replicas. In our proposed framework, update propagations are carried out through pull and push based approaches. Push approach enforces consistency model on primary copies of DARs located in a central MySQL database. In this model; whenever updates occurred on a primary copy of a DAR, they are being propagated immediately to each integrated annotation tool to update DARs on their site. Push approach is handled via Digital Entity Update Management module discussed in 3.7.3.2. However, pull approach is a time-based consistency control approach [154]. We are periodically checking DARs from each annotation tool for any updates. If there is any, then we are first pulling them out. Next, we are applying them onto the primary copy of each DAR, which is stored in a relational database (MySQL) with additional metadata. Concurrent updates on a shared document are handled based on our concurrent access policy defined in Section 4.4.1. We have also designed a rollback structure to help for consistency maintenance. It basically allows users to rollback to a previous state at any time. The rollback mechanism explained in detail in Section 3.7.3.4. Figure 3-4 represents the overall architecture of our proposed consistency framework. Explanation of the architecture components are given in Section 4.4.2 in detail.

4.4.1 Duplicate Detection and Handling Concurrent Updates

In our proposed CFDAR, it is crucial to identify if two records are representing the same document or not. During the consistency maintenance process, which is utilized periodically in the background by the Annotation Tools Update Manager module as explained in Section 4.4.2.3; each record is retrieved from annotation tools, and processed to form a digital entity as explained in Communication Manager in Section 4.4.2.2. We use our duplicate detection algorithm as explained in Section 3.4 to find a DE's matching primary copy if it exists in the database. According to our duplicate algorithm result, if a digital entity does not exist in the database, then this is treated as a new update and it is passed to the DE Update Management module to be inserted into the system as a major event, and propagated to all replicas stored at the integrated annotation tools by DE Entity Update Management module. If we can find a matching primary copy for this digital entity, then our Annotation Tools Update Manager compares these two digital entities to identify that whether any metadata field of the digital entity is updated or not. If there are any available updates, then they are processed by Digital entity Update Management module to push the updates to the annotation tools as explained in Section 3.7.3.2.

The main goal of concurrency control is to allow processes to work on a shared data simultaneously in a way that the shared data left in a consistent state after modifications. The consistency is maintained by giving processes access to shared data in a specific order in which the final result is the same as if all processes had run sequentially. Concurrency control algorithms can be classified as pessimistic and optimistic approaches [110]. Pessimistic concurrency control algorithms assume that

conflict happens frequently and they block (synchronize) access to the data item until it is upto date. Pessimistic concurrency control algorithms can generally be categorized based on how the read and write operations are synchronized. Synchronization can be provided through either mutual exclusion or ordering processes by using timestamp values [110]. Optimistic algorithms, on the other hand, assume that conflicts are rare and allow users to access data (without synchronization), and work on their private copies independent from each other. The conflicts are resolved after they occurred in optimistic algorithms. In our proposed system there could be two types of conflicts:

a. *Overlapping Conflicts*: If two updates are updating the same metadata field of a same DE, then this kind of conflicts are counted as *Overlapping Conflicts*. They are not automatically resolved and logged into the system so that they can be resolved by users.

b. Non-overlapping Conflicts: If two updates are modifying the different metadata field of a same DE, then this type of conflicts are called *Non-overlapping Conflicts*. These types of conflicts can be resolved automatically by the system unless defined in the property file to be handled manually.

Based on [124], elements of our optimistic replication approach can be defined as:

- *Object, Replicas, and Sites.* Our minimum unit of replication is the records located at annotation tools and a central MySQL database. A minimum unit is called “Object”, which is defined as a Digital Entity (DE) in our proposed architecture. A copy of an object is called “Replica”. Replicas can be stored as read-only or writeble replicas in sites. Sites that can update a replica are called

master-sites. All of our sites that are the integrated annotation tools and the main system are master-sites.

- *Operations.* Operation is a self-contained update to a replica. In our proposed thesis, each operation is an event. Operations are propagated to replicas and applied there in the background. In our proposed CFDAR, operations might come from annotation tools or from our main system. When a record updated in main system, then this update is applied in the main system locally, and propagated to the integrated annotation tools in the background to be applied there. Our system guarantees that states of the replicas will converge eventually.
- *Propagation.* Any update happened in the main system is logged as an event so that it can be propagated to the replicas in the background to be applied there. Usually epidemic propagation is used for propagation mechanism. However, our integrated annotation tools are external tools and we do not have control over them, hence we assume that the integrated annotation tools receive the propagated updates by our push mechanism.
- *Tentative Execution and Scheduling.* When updates are propagated in the background, all sites may not have the updates in the same order. Hence, each site uses an appropriate ordering mechanism to order the coming updates so that the final results are consistent with other sites. In our CFDAR, we assume that updates happened in our main system and propagated to the integrated annotation tools are arrive in the same order that they are created. Updates took place in annotation tools and collected via Annotation Tools Update Manager explained in 4.4.2.3.

- *Detecting and Resolving Conflicts.* With no synchronization access to data, multiple users can modify the same record concurrently. Conflicts should be detected and resolved. Conflict resolution is depent on the application. Some systems resolve conflicts automatically and some just marks them so that users can fix them. In our CFDAR, conflicts are resolved based on the defined policy in the properties file. Our conflict resolution can be set to "turn-on", or "turn-off". If conflict resolution is set to "turn-on", then non-overlapping conflicts are resolved automatically and logged into the system, whereas overlapping conflicts are not resolved but they are logged into the database so that users can be notified to resolve them later. If conflict resolution is set to "turn-off", then all conflicts are just logged into the system to be resolved by users manually.
- *Commitment.* "Commitment refers to an algorithm to converge the state of replicas by letting sites agree on the set of applied operations, their final ordering, and conflict resolution results" [124]. We assume that all the integrated annotation tools and the main system agree on the set of applied updates, their final ordering, and conflict resolutions due to the external nature of the integrated annotation tools.

In our proposed CFDAR, it is very rare to have concurrent updates on a same document due to the asynchronous nature of our proposed system. However, our policy for concurrent updates is to adopt optimistic replication approach. The major goal of any optimistic replication system is to maintain consistency between the replicas, and usually optimistic replication systems ensure eventual consistency. In this approach, we have selected the following optimistic replication design choices according to [124]:

- *Number of writers.* There are single-master and multi-master systems. Single-master systems select one replica as master and all updates carried out through the master replica and disseminated to other replicas, whereas multi-master systems allow updates to take place at multi replicas and propagate updates in the background. Our proposed system is a multi-master system since all records can be updated at all annotation tools indepently. Then, updates are collected and propagated to main system and all integrated annotation tools (replicas) in the background periodically.
- *Definition of operations.* There exist two choices for definition of operations: (a) State Transfer; (b) Operation Transfer. State transfer systems propagate only the content of the update to the replicas while operation transfer systems define operations more semantically. In our design, we have selected to use state transfer due to the nature of the integrated annotation tools.
- *Scheduling.* The main purpose of the scheduling is to schedule operations on replicas so that their states are consistent with each other. Scheduling can be categorized as: (a) Syntactic; and (b) Semantic. "Syntactic scheduling sorts operations based only on information about when, where, and by whom operations were submitted. Timestamp-based ordering is the most popular example. Semantic scheduling exploits semantic properties, such as commutativity or idempotency of operations, to reduce conflicts or the frequency of roll-back. Semantic scheduling is used only in operation-transfer systems, since state-transfer systems are oblivious to operation semantics by nature" [124]. We have selected to use syntactic scheduling via the use of the defined priority of the

integrated annotation tools values in our main system. We can still use the time-stamp values in our system to sort the updates in the history of each record.

- *Handling Conflicts.* Conflicts are inevitable when there is a concurrent access to data. Pessimistic replication approaches try to prevent conflicts before they happen by synchronizing access to the data. Some systems ignore conflicts and somes try to resolve them after they occurred. In our proposed system, we resolve the conflicts based on the conflict types and the defined policies on the properties file.
- *Propagation Strategies and Topologies.* Each local update must be transferred and applied on all replicas in order to ensure consistency. Each replica need to log their modifications when they are offline, and need to decide when to communicate and exchange updates with other replicas. "Propagation policies can be classified along two axes, communication topology and the degree of synchrony" [124] as illustrated in Figure 4-2. According to Figure 4-2, "The degree of synchrony shows the speed and frequency by which sites communicate and exchange operations" [124].

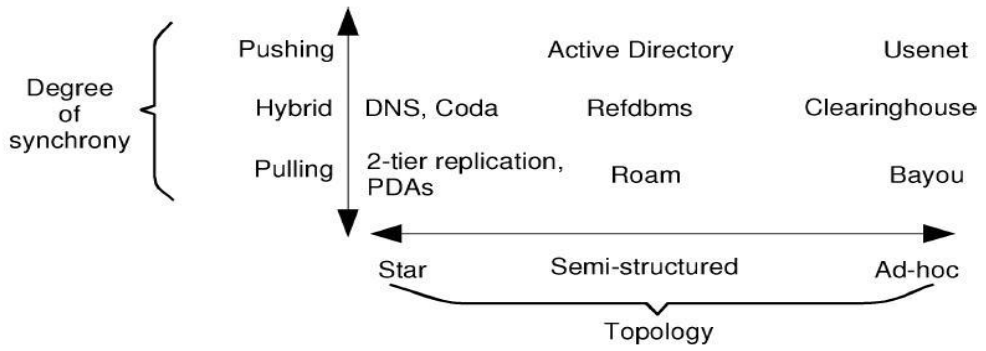

Figure 4-2: Design choices regarding operation propagation (image is taken from [124]).

Pull-based systems work based on the idea that each site periodically or manually checks the other sites such as PDAs for manually check and DNS for periodic checks for new operations or updates. In Push-based systems, however, a site with new updates propagates its updates to other sites. In our proposed system, we have combined pull and push based update propagation strategies.

- *Consistency Guarantees.* Consistency guarantees vary from single-copy consistency to eventual consistency based on the divergence of the states of replicas in optimistic replication systems depicted in Figure 4-3.

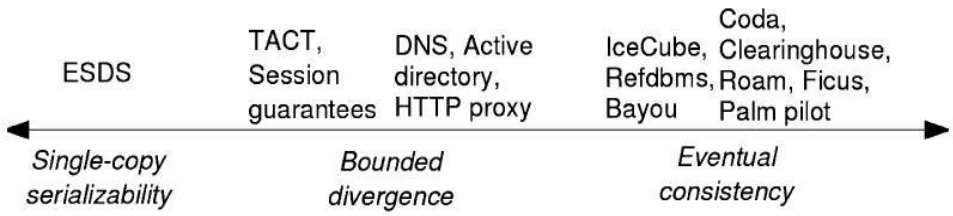

Figure 4-3: Choices regarding consistency guarantees (image is taken from [124]).

In our consistency framework, our system guarantees eventual consistency among replicas by adopting the optimistic replication approach.

In our proposed research, concurrent updates occurrences are very seldom, and they may occur when:

I. A user uses an annotation tool's own UI to update a replica record and another user concurrently tries to update the same copy through its primary copy by using Digital Entity Manager of the proposed system.

II. Two or more users try to update a record through the primary copy by using Digital Entity Manager of the proposed system.

III. While consistency manager is working in the background to collect and process updates on primary copies and replicas, another user tries to update a record via its primary copy by using Digital Entity Manager of the proposed system.

In the first case, we do not have much control how the annotation tool handles the coming concurrent update requests. In order to handle inconsistencies, annotation tools are supposed to notify our system once an update occurs in their system, but they do not have such functionality. Moreover, they are independent (external) systems and we cannot lock them during the update executions to prevent inconsistencies. For example; let's assume that an update ($Update_{primary}$) coming from the replica's primary copy is executed before a user's update ($Update_{annotation}$) coming from the annotation tool's UI. Then $Update_{primary}$ will be lost. Since, $Update_{annotation}$ overwrites $Update_{primary}$. Let's think the other case that $Update_{annotation}$ is executed before $Update_{primary}$. Then, $Update_{annotation}$ will disappear due to the replacement of it by $Update_{primary}$. So, above cases are exceptions in real life domain as explained in 4.3. In the second and third cases; users get a copy of records to their private work space and work on their records. After they finish their modifications, then their updates are committed into the system if the same records have not been modified yet or there are not any conflicts. If there are conflicts, then the conflicts are resolved based on the defined policy in the properties file and the users are notified about these conflicts.

4.4.2 Overview of the Architecture Components

Our CFDAR modules can be placed under four umbrellas: Annotation Tools, Communication Manager, Annotation Tools Update Manager, and Digital Entity

Manager. The consistency framework uses Timestamp Generator, and Data Modules explained in Section 3.7. The detailed explanation of the CFDAR modules and their responsibilities are explained in the following sections respectively.

4.4.2.1 Annotation Tools

Annotation tools represent the unified and federated annotation tools into our proposed CFDAR. They hold data and metadata about documents in their systems. Another word, they are distributed repositories that stores data and metadata in their system with provided services. The common feature of these annotation tools is that they allow their users to tag their content and provides different services for managing their resources. For instance; Figure 4-4 depicts popular tags on del.icio.us, and users of these systems can access related documents or similar research groups via the tags. These systems are discussed in detail in Section 2.1.

In our framework, we intend to use these tools as replicas to store metadata of scientific documents. The records in these tools called DARs are the replica copies of the records stored in the MySQL system database. As we explain earlier in CHAPTER 3, we represent several data and metadata coming from various sources as events in our event-based infrastructure. The documents represented and stored in the MySQL system database as events are the primary copies of the DARs located at the integrated annotation tools. Main responsibility of our CFDAR is to maintain a consistency between replica copies of data and metadata located in integrated annotation tools and their primary copies stored in a MySQL system database.

Figure 4-4: Popular Tags in del.icio.us

4.4.2.2 Communication Manager

Communication manager depicted in Figure 4-5 transports the data between the computing nodes. It is responsible for uploading or downloading data from annotation tools through their defined gateways. It retrieves the records from annotation tools via HTTPClient native libraries by using either: (1) Annotation tool's API and get the response in XML format. Records are then parsed by using a DOM parser and XPATH [155]; or (2) HTTP GET, and POST method resulting in getting the response in RSS or HTML format. In RSS type responses, documents are parsed by using a DOM parser and XPATH, and in HTML type responses, data is parsed after cleaning faulty HTML by using JTidy [156] native libraries. Having retrieved and parsed documents, Communication Manager passes the mined data to Annotation Tools Update Manager explained in detail in Section 4.4.2.3. Updates are disseminated to annotation tools by Digital Entity Update Management module described in Section 3.7.3.2 via: (1) Annotation tools API; or (2) HTTP GET, POST methods through HTTPClient [130]

native library unless an annotation tool provides an API. Communication Manager's modules are explained in the following sections respectively.

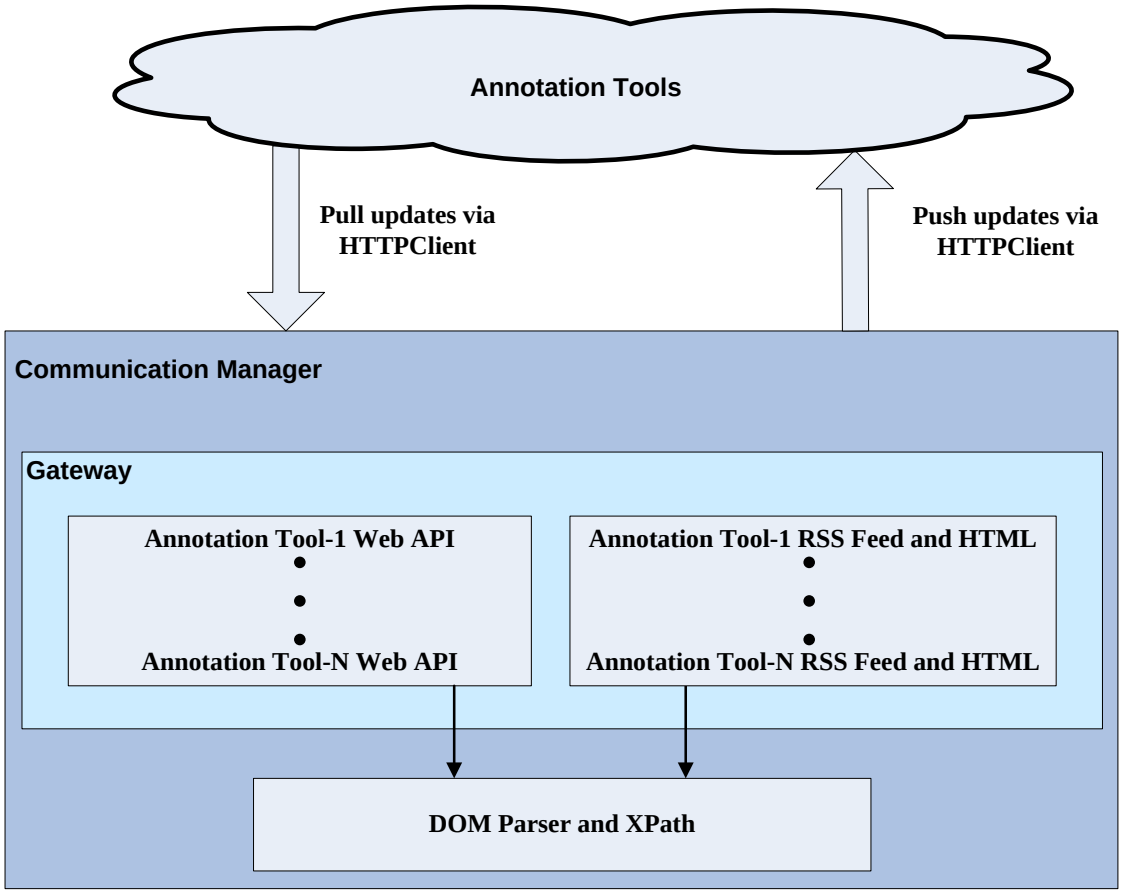

Figure 4-5: CFDAR Communication Manager

4.4.2.2.1 Gateway

Gateway represented in Figure 4-6 is an interface between the CFDAR and annotation tools. It is both entrance and exit point for the incoming and outgoing messages. Our proposed CFDAR communicates with annotation tools through their gateways. Another word is that Gateway connects Consistency Framework to the annotation tools by using libraries and tools of those native environments.

Each annotation tool provides their clients with various ways to interact with their system: (1) Web API allows clients to retrieve, modify and post data easily to an annotation tool; (2) RSS allows clients to retrieve data easily from an annotation tool. However, to modify an existing DAR or to post a new one; HTPP GET and POST methods need to be used via HTTPClient native libraries. The communications are carried out through HTTP methods by using HTTPClient native libraries [130]. An individual gateway is created for each interacting annotation tool, which has its own communication structures.

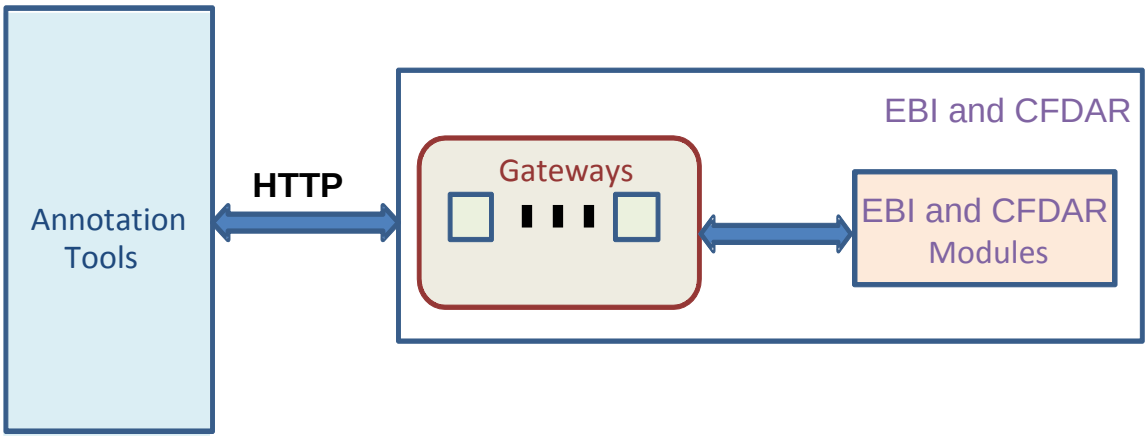

Figure 4-6: Gateway

4.4.2.2.2 Parser

Parser is a native library used for parsing the responses coming from annotation tools. There are several parsers to utilize in XML processing. DOM parser is the most widely used one. It reads and validates the XML documents. If the document is valid, then it returns a document object tree. We can randomly access any element since each element is entirely kept in memory. As a result, it provides a very efficient navigation mechanism over the parsed document. On the other hand, its drawback is that it requires

large amount of memory in order to hold the whole parsed document as discussed in detail in Section 2.4.3. Most of the major annotation tools provide their Web API so that users can communicate with their services easily. In our prototype implementation (described in CHAPTER 5), we have used JDOM [157] parser as our parsing library. In some annotation websites, they do not provide a Web API for their services. In order to communicate with those annotation tools, we have used XPATH to retrieve the desired element of the document and JTidy native libary [156], which is used for cleaning faulty HTML and provide a DOM interface to the documents that is going to be parsed.

4.4.2.2.3 Web API

Web API (Application Programming Interface) is a service for accessing data on annotation tools. Most of the major annotation tools provide their Web API and RSS feeds for an easy access to their data. Their Web API and RSS feed return a document in XML format, which can be parsed easily by using a DOM parser, to the requester.

Document kept at annotation tools can be retrieved, modified via either their Web API or HTTP (POST and GET) methods through HTTPClient tool by passing the necessary parameter to HTTPClient object. Having executed their Web API for posting or editing a document, we are going to receive a response in XML format similar to the one displayed in Figure 4-7.

```
<posts tag="" user="user">
  <post href="http://www.weather.com/" description="weather.com"
  hash="6cfedbe75f413c56b6ce79e6fa102aba" tag="weather reference"
  time="2005-11-29T20:30:47Z" />
  <post href="http://www.nytimes.com/"
  description="The New York Times - Breaking News, World News &
Multimedia"
  extended="requires login" hash="ca1e6357399774951eed4628d69eb84b"
  tag="news media" time="2005-11-29T20:30:05Z" />
</posts>
```

Figure 4-7: Web API Response

However, the response is going to be in HTML format for requested operations in HTTP methods instead of XML type format.

4.4.2.3 Annotation Tools Update Manager

CFDAR consists of pull and push based approaches to propagate the updates to the replicas. Annotation Tools Update Manager depicted in Figure 4-8 utilizes pull based approach to obtain the updates happened on annotation tools, whereas Digital Entity Update Management module explained in Section 3.7.3.2 utilizes push based approach to propagate the updates made on primary copies via the proposed system services to the integrated annotation tools. In pull based approach, Annotation Tools Update Manager utilizes handlers to retrieve DARs periodically from annotation tools introduced in Section 4.4.2.1 via Communication Manager described in Section 4.4.2.2 to collect updates occurred on annotation tools. Next, it passes the updates to Digital Entity Update Management module explained in Section 3.7.3.2 so that updates can be disseminated to the integrated annotation tools immediately after being applied on the primary copy of each DAR (Push based approach). Annotation Tools Update Manager's main responsibilities are: (1) Obtaining DARs from annotation tools regularly via

Communication Manager; (2) Determining the updates by comparing DARs and their primary copies stored in MySQL system database; (3) Passing updates to Digital Entity Update Management module which has a embedded module to propagate updates to the integrated annotation tools to reflect the changes on DARs.

Annotation Tools Update Manager uses a handler such as Java thread for retrieving DARs from each annotation tools regularly as it is defined in the properties file. Another word is that it obtains records from each annotation tool and finds out the updates periodically. If there are any conflicting updates that are coming from different annotation tools for a same document, Annotation Tools Update Manager processes the updates as described in Section 4.3 and Section 4.4.1.

Collecting updates from documents that are coming from Communication Manager requires: (1) Finding the primary copy of each replica record by using duplicate detection algorithm discussed in Section 3.4; (2) Comparing each replica record with its primary copy to figure out modifications if there is any. After indentifying the updates, next step is to pass them to Digital Entitiy Update Management module so that updates can be applied on their primary copies and disseminated to replicas located at annotation tools. Integrated annotation tools do not support publish-subscribe mechanism forcing Digital Entitiy Update Management module to use unicast communication to propagate updates to replicas. However, any application that require and support publish-subscribe mechanism, then broker address and topic can be defined in a property file to provide updates via publish-subscribe mechanism by connecting to the broker and subscribing a topic.

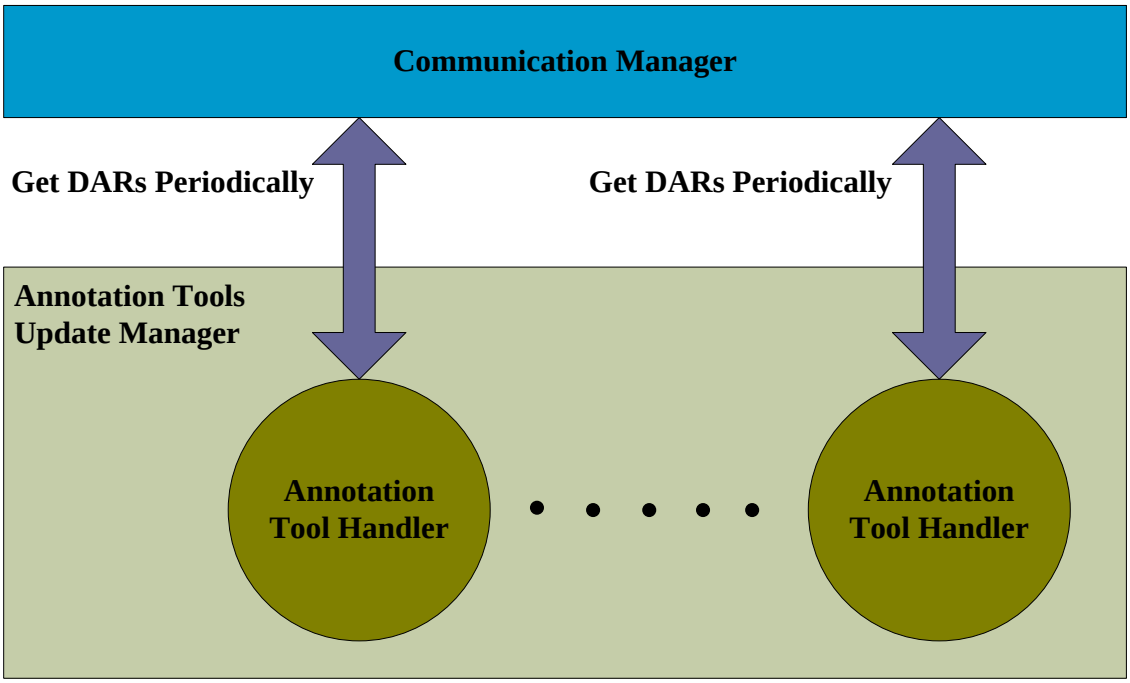

Figure 4-8: Annotation Tools Update Manager

4.4.2.4 Digital Entity Manager

Digital Entity Manager is an umbrella name for a group of modules that contributes to DAR management together. Its modules are: (1) Digital Entity Update Management; (2) History and Rollback Management; (3) Periodic Updates Management; (4) Events and Dataset management.

Digital Entity Update Management module is used by CFDAR. In Digital Entity Update Management module explained in Section 3.7.3.2, update propagations are carried out through push based approach. It applies modifications on their primary copies and disseminates the updates immediately to their replica copies kept at integrated annotation tools. Figure 3-5 displays the Digital Entity Manager and its components. Detailed explanation of its modules can be found in Section 3.7.3.

4.5 Summary

In this chapter, we explained CFDAR and its modules. Our CFDAR has a moduler architecture which improves the maintanence and simplicity of the system. The modules can be classified into four sub-groups. The first group that contains the annotation tools where DARs are located. The second group is responsible for transporting the data between computing nodes. The third group is in charge determining the updates mades through the annotation tools. Finally, the last one is to handle the updates made on to a primary copy of a DAR.

CHAPTER 5

The Prototype Implementation of Event-based Infrastructure and Consistency Framework

This chapter represents implementation details of a prototype of the system architectures mentioned in the earlier chapters. To demonstrate the effectiveness and applicability of CHAPTER 3 and CHAPTER 4, we have implemented a prototype service enabled framework based on these architectures. The prototype system is implemented by utilizing following technologies and open source tools: (a) Java 2 SDK, Standart Edition with version 1.5 [158], (b) Apache Axis Web Service Platform with version 1.2 [7]; (c) Apache Tomcat Servlet Container with version 5.0.28 [8]; (d) HttpClient Technology with version 3.0.1 [130]; (e) JTidy Tool with version 04aug2000r7 [156]; (f) JDOM with version 1.1 [157]. Our implementation called Internet

Documentation and Integration of Metadata (IDIOM) is an open-source and available from [159].

In this section, we discuss the implementation details of Event-based Infrastructure and Consistency Framework for Distributed Annotation Records (CFDAR) prototype system. First, we review the prototype IDIOM implementation. Second, we discuss the Event-based Infrastructure of the IDIOM system. Finally, we discuss Consistency Framework for Distributed Annotation Records of the IDIOM system.

5.1 IDIOM System Implementation Overview

IDIOM provides a collaborative Cyberinfrastructure based scientific research environment [2, 160]. Its tools and services are backed by a MySQL database which store user and community specific data and metadata and is configured into three applications: (1) A model for scientific research which links both traditional simulations and observational analysis to the data mining of existing scientific documents; (2) A model for a journal web site supporting both readers and the editorial function; (3) A model for a natural collection of related documents such as those of a research group or those of a conference.

Figure 5-1 shows the overall architecture of the prototype IDIOM system. This system consists of five main layers: (a) the *client* layer; (b) the s*ervice* layer; (c) the *server* layer; (d) the *helper* layer; and (e) the *data* layer. The client layer of the IDIOM system is made up of Java Server Pages (JSP) [43], which is translated into servlets by an Apache Tomcat J2EE Web container and generates dynamic content for the browser. The service layer provides interfaces to access the IDIOM systems's Web Services, and the

client layer communicates with the Server layer over the HTTP protocol through SOAP messages encapsulating WSDL-formatted objects. The Server layer consists of several

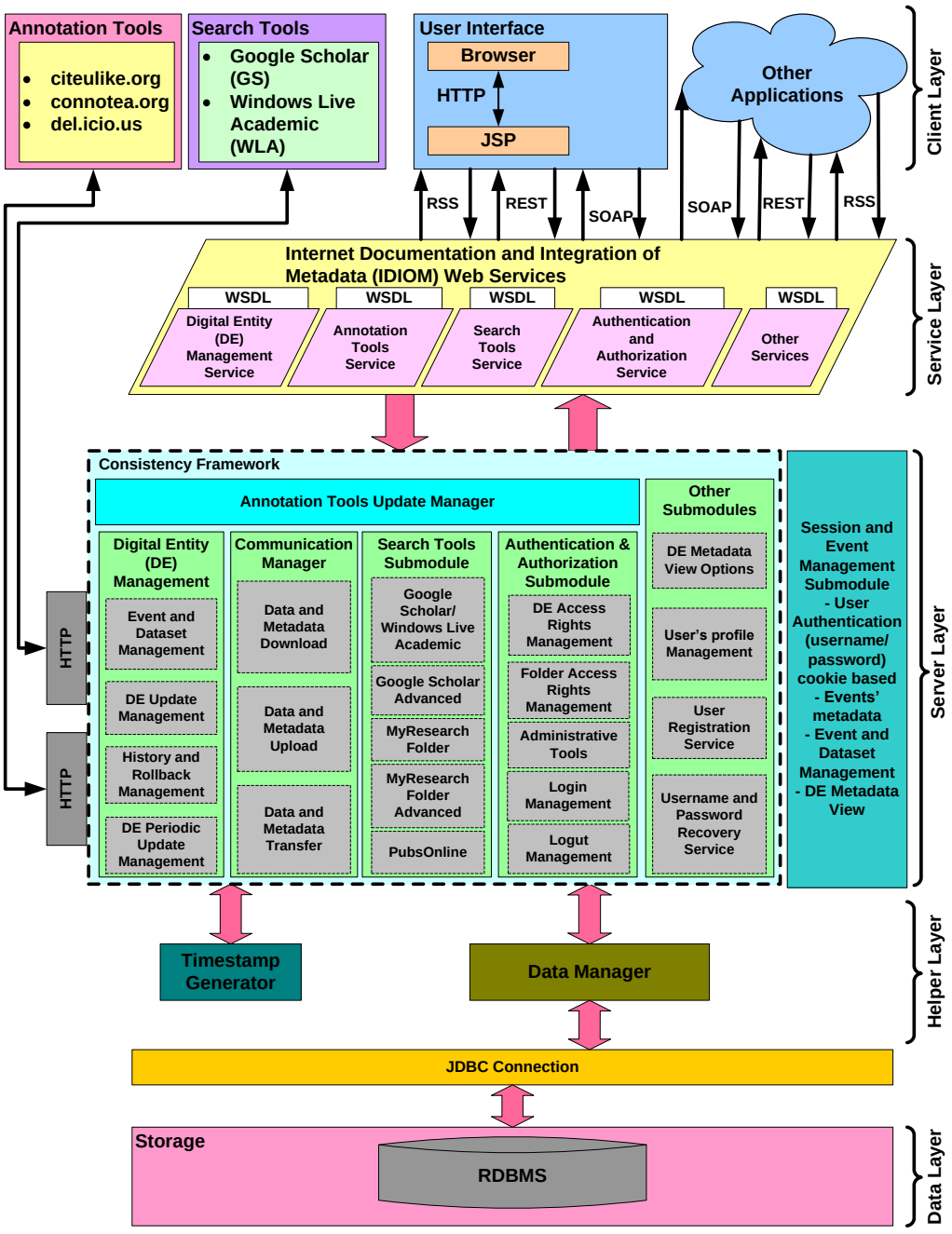

Figure 5-1: Internet Documentation and Integration of Metadata (IDIOM) Architecture

modules that constitute the main architecture blocks of the IDIOM system to handle the coming requests from the service layer. The helper layer provides synchronized timestamp values and handles the requests to be forwarded to Data Manager so that it can communicate with the data layer through JDBC connection. Finally, the data layer is composed of a MySQL system database.

We have followed Web 2.0 design patterns [161] in designing the IDIOM system. Below, we list these patterns and discuss how they were applied in designing IDIOM:

Delivering services, not packaged software: IDIOM is a collection of tools and services that can be accessed over the Web (either through a user interface or programmatically through Web services). It will evolve by introducing new features; still its users won't have to install new versions of the software.

Producing hard-to-recreate data that gets richer as more people use the system: By combining data from a variety of sources, IDIOM will create added-value data and metadata generated with specific communities in mind. As more people participate in a community, the collection of the data and metadata managed by that community will increase in quantity, leading to the potential for improved precision of the automated system tools.

Harnessing collective intelligence: Through its integration with the social bookmarking tools, IDIOM can leverage data and metadata from a large number of researchers. Moreover, the system can handle both individual users and groups of users, and supports sharing and collaboration between group members.

Leveraging the long tail through customer self-service: The term "long tail" here refers to the concept formulated by Anderson [162] that non-hit products can collectively

make up a market share that may exceed the relatively few current hits, bestsellers or blockbusters, provided the store or distribution channel is large enough (this business model is leveraged for example by Netflix or Amazon.com). IDIOM aims to support research communities, such as the members of a research project, a group interested in a particular chemical compound and so on, by allowing them to create system accounts and to use the community-building tools for their specific usage scenarios.

Software above the level of a single device: Currently, the IDIOM user interface runs in a browser. However, because of its layered design and the use of J2EE technology, system front-ends for other devices, such as PDAs, can be developed at low cost. In addition to these design patterns, we have followed two general principles: (a) every component is packaged as a service as long as this packaging does not imply an unacceptable performance degradation; b) if a needed capability exists and works well but is insufficient in some fashion, we try not to replace it but rather wrap it as a service so we can interact with its natural interface but easily input and output information through its service interface.

5.2 Event-based Infrastructure

In our proposed Event-based Infrastructure; all documents, their metadata and modifications to documents are represented as events (major or minor). Events are used to keep track of changes to documents and metadata. The main aspects of the implemented Event-based Infrastructure of the IDIOM system can be enumerated as follow:

- *Digital Entity (DE):* It is a collection of metadata that represents metadata fields of a scholarly publication. The metadata fields of a DE in our implementation of the IDIOM system are displayed in Figure 5-2.

Figure 5-2: The Content of a Digital Entity

- *Major Events*: Each entry of a digital entity in to the system or deletion of an existing digital entity from the system is considered as a major event. A major event can have as many as minor events related to it. If a major event does not have any minor events, then the DE data consists of only the major event data.
- *Minor Events*: Every updates to an existing digital entity in the system are defined as minor events in our proposed architecture. Therefore, minor events represent the modifications to an existing DE in the proposed system. During the process of building a DE, minor events are processed on top of the major event by their timestamp.

- *Dataset*: A dataset is a collection of minor events in our proposed architecture. Each update represents a state of a DE, and we provide a mechanism to go to any state of a DE in its history. A dataset allows combining several updates into one state in our Event-based Infrastructure.
- *Rollback mechanism*: We have designed a rollback mechanism based on the concept of events and datasets. It allows going back and forward to any state on the history of a DE.
- *Users and Profiles:* The IDIOM system supports individual users and groups of users. Users' personal information and the login information for bookmarking web sites are accessible through the user's profile. More specifically, user's profile contains the system password, email address, full name, login information for annotation web sites (citeulike.org, connotea.org and del.icio.us), and the group membership information. Users can access and modify their profile settings at any time; while logged in users can: (a) Change their system password; (b) Update their profile including the full name, email address and the username and password for the annotation web sites; (c) Make requests to subscribe to any available group. For each DE, there are three types of access rights: Read access right, Write access right, and Delete access right. Users who have Read access for a DE can read that citation. Only users who have write access for a DE can update that citation. Delete access is required for deleting DEs. These access rights are defined with respect to three kinds of users: Owner who is the user that initiates the citation metadata creation; Group which is the group to which the owner belongs; other users. There is only one owner of a citation record. However, there

might be more than one group for a citation. The owner of a citation record can specify the citation rights for all three kinds of users mentioned above.

- *User Session:* Due to the stateless nature of HTTP, a number of alternative mechanisms have been developed for applications that need to maintain a conversational state. The HTTP session API, which is a component of the Java Servlet specification, provides a mechanism for web-based applications to maintain a user's state information. This mechanism, which is called *session*, is usually associated with a user and supports the management of the user's state information on the server side. A session is represented by an HttpSession object, which stores and provides access to the user specific data. In the IDIOM system, the user's session is instantiated once a user logs into the system. The session can be later accessed through the JSP pages. All requests to get MoreInfo on a record or to update a record bring a copy of the original record into the user session, which is a private workspace for the user.
- *Messaging Format:* Provided services of the IDIOM system communicate with its clients via exchanging messages in XML format. The schema for the content of a DE is depicted in Figure 5-3:

element **digitalentity**

diagram

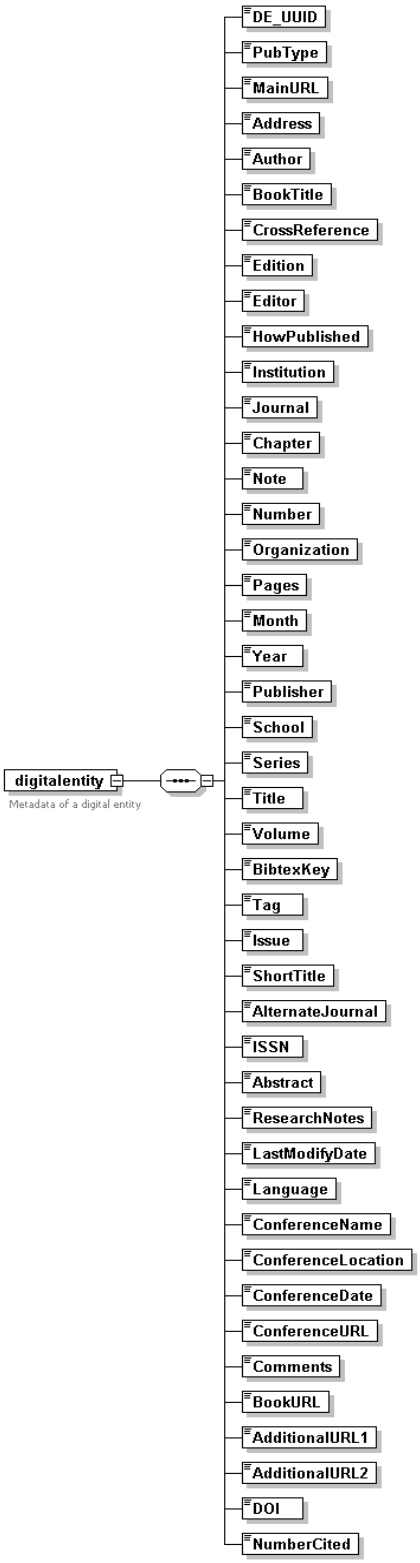

Figure 5-3: Schema of DE Content

- *Confirmation of Service Execution and Exception handling:* IDIOM services are deployed by using Axis 1.2 Web Service technology under Apache Tomcat Container. Confirmation of requested services is returned to the clients in XML format. Furthermore, any exceptions occurred during the execution of services such as originating from integrated annotation tools or related to service implementations etc. are caught and a confirmation message is returned to clients in XML format.

Finally, modules of the prototype IDIOM system can be categorized as: (1) Annotation Tools; (2) IDIOM Web Services; (3) Session and Event Management; (4) Digital Entity Management; (5) Search Tools; (6) Authentication and Authorization; (7) Other; (8) Timestamp Generator; (9) Data Manager. In the following sub-sections, we give a brief description of the functionality provided by each module.

5.2.1 Annotation Tools

Annotation Tools are the integrated annotation tools into the IDIOM system to store replica copies of the primary copies referred as DE stored in a MySQL system database. It implements the Annotation Tools abstract layer as described in 4.4.2.1. The records kept at annotation tools called DARs can be accessed via IDIOM system services and user interfaces. Users can upload records from repository to these tools, download records from these tools into a repository, or transfer records between the integrated annotation tools. In the current implementation, IDIOM system unifies and federates Connotea, CiteULike, and Delicious tools.

5.2.2 IDIOM Web Services

IDIOM Web Services implements the Event-based Infrastructure Services abstraction layer as explained in Section 3.7.2. IDIOM Web Services provide access to modules and their services via SOAP calls over HTTP protocol in current implementation. The IDIOM Web Services can be accessed via different protocols through the supported interfaces as well.

5.2.3 Session and Event Management Module

The goal of this module is to store user specific data such as cookie-based user credentials (password/username), modifications to a DE as minor events, and the "view options", which control the level of detail with respect to the metadata fields displayed for each DE, into users' session. A session is a user's state information, and maintained on the server side [163]. From the moment user logged in the IDIOM system, user credentials, any changes made to a DE, and view options for metadata fields of a DE are all saved in the user session. It also serves as a private workspace for the user and users can concurrently modify their copy of records. When a user logs out from the IDIOM system, all unused minor events (modifications to a DE) for a dataset creation are removed.

The updates to DEs are saved into the logged user's session and they can be accessed via the implemented Event Management Module user interfaces. Event Management Module user interfaces allow users to access and simulate the minor events, which represents the updates for a digital entity, before creating a dataset(s) by selecting available minor events for a digital entity. Another word is that users can review their updates called minor events, create datasets, simulate their updates on the DE, and

confirm their updates if they wish. Once users confirm the updates, then these updates inserted into MySQL database for the related DE as minor events and removed from the logged user's session. Upon a request to access the DE metadata, these minor events are automatically processed on top of the major event of the DE to build the DE metadata based on the selected path. The selected path could be a user's events, or a group's events, or all events belong to this DE as a default option. Figure 5-4, Figure 5-5, and Figure 5-6 displays the available paths to process events to build a DE, content of a minor event, and application of the minor event on the DE.

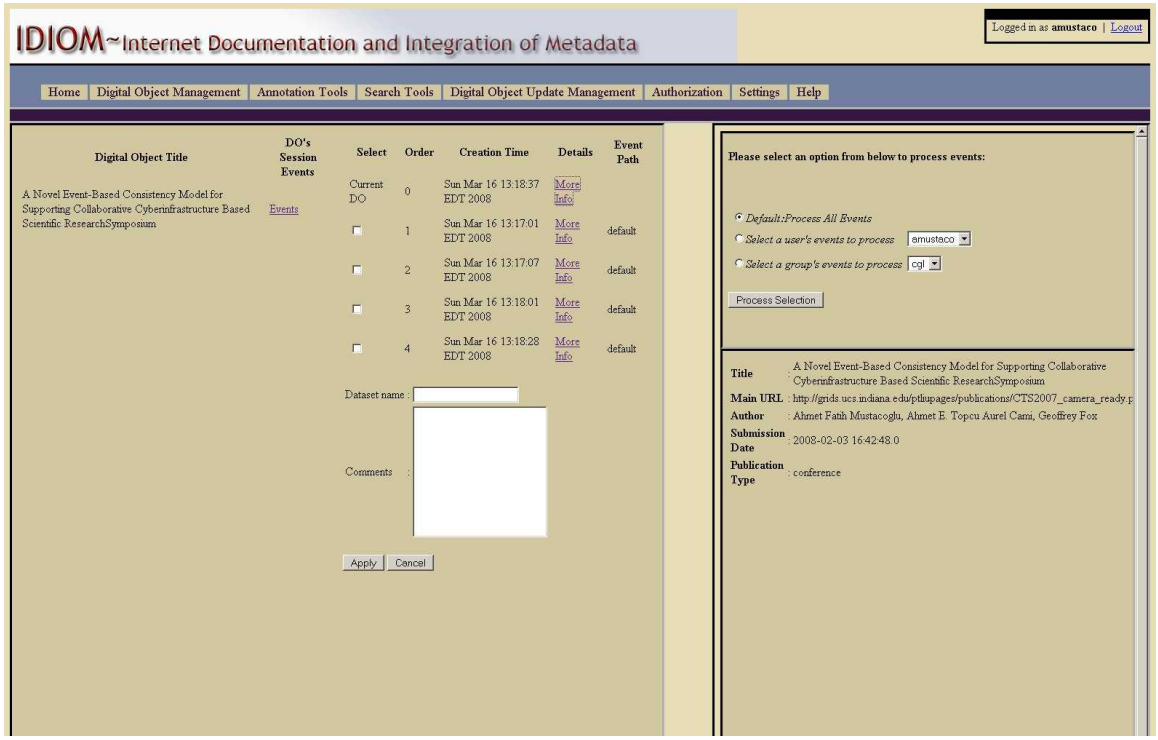

Figure 5-4: Current Metadata of a DE

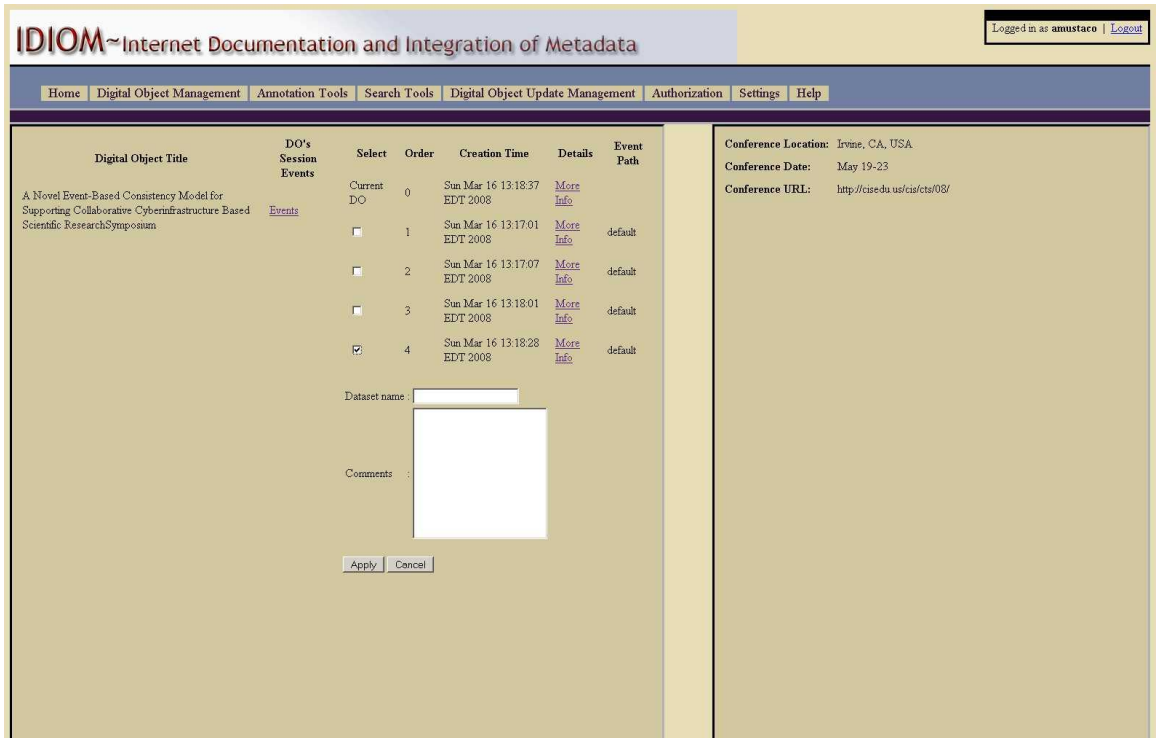

Figure 5-5: Content of a Minor Event

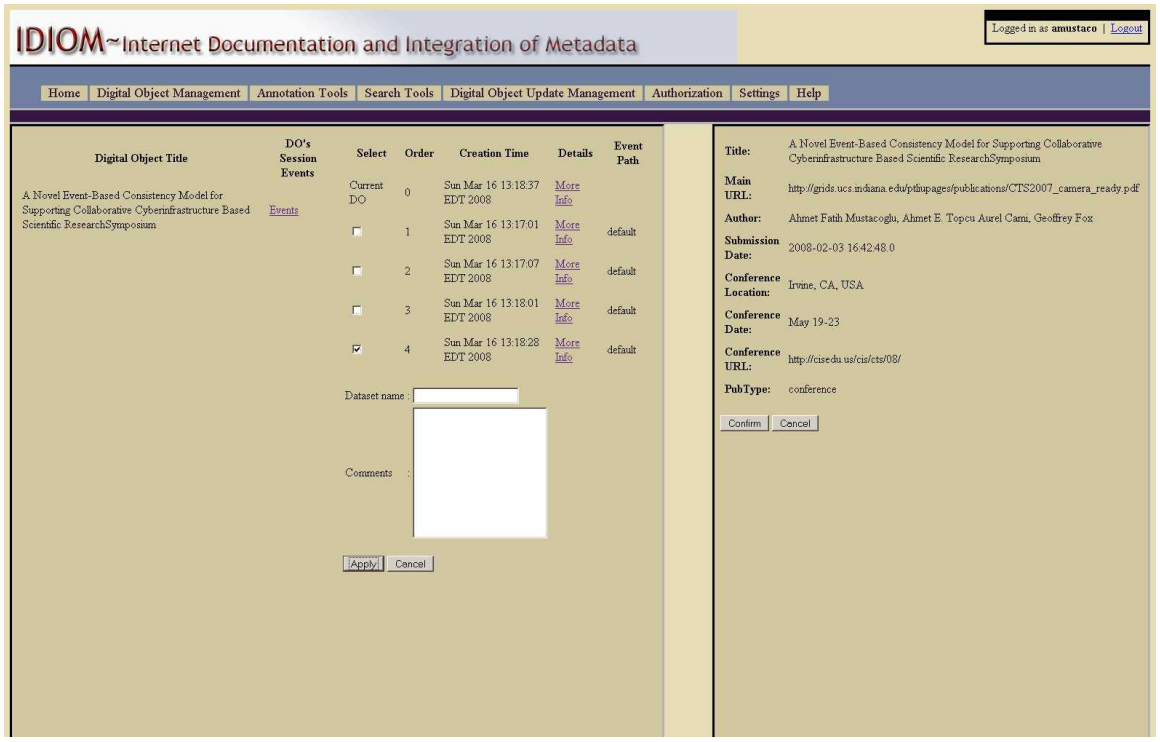

Figure 5-6: Application of a Selected Minor Event to a DE

5.2.4 Digital Entity Management Module

Digital Entity Management module is responsible for: (1) Providing a service for inserting a new DE into the IDIOM system, and push the new entry to the integrated annotation tools via Communication Manager described in Section 5.3.2.; (2) Implementing the Events and Dataset Management services, and providing a service to view detailed information about a DE by utilizing Event Processing Engine (Implements Event and Dataset Management abstraction layer explained in Section 3.7.3.1); (3) Providing services for updating an existing DE, and it utilizes push-based consistency maintenance approach by pushing the updates immediately after they occur to the integrated annotation tools via Communication Manager described in Section 5.3.2. (Implements Digital Entity Update Management abstraction layer discussed in 3.7.3.2); (4) Providing an access to the history of a DE and rollback mechanism, from its entry into IDIOM system to present (Implements History and Rollback Management abstraction layer as discussed in Section 3.7.3.4); (5) Providing a service to retrieve and apply updates belonging to other users on their DEs by Periodic Update Management service (Implement Periodic Update Management abstraction layer as introduced in Section 3.7.3.3).

New DE service deals with the requests to make and insert a new digital entity into the IDIOM system. Created new entity is saved as a major event into IDIOM system's main database. If the entity about to be created already exists in the IDIOM system, then the request is forwarded to DE Update Management module to be handled.

Event and Dataset Management service handles with creating events (minor and major) and datasets for the related DEs. In the current implementation of the IDIOM

system, minor events are kept in the logged user's session. Once user created datasets from the minor events, then they are sent to this module to be processed. Furthermore, coming requests for a new DE entry is represented by creating a major event for it by this module. Event and Dataset Management module also provides a *More Info* service, which implements the concept of building DE metadata from its events. More Info service is achieved by processing the selected minor events that are ordered by their timestamp on top of the initial metadata of a DE (major event of it) by Event Processing Engine. The selection of minor events can be based on a user, a group, or a default selection that includes all the minor events for a DE. The implemented user interface to use More Info service is depicted respectively in Figure 5-7.

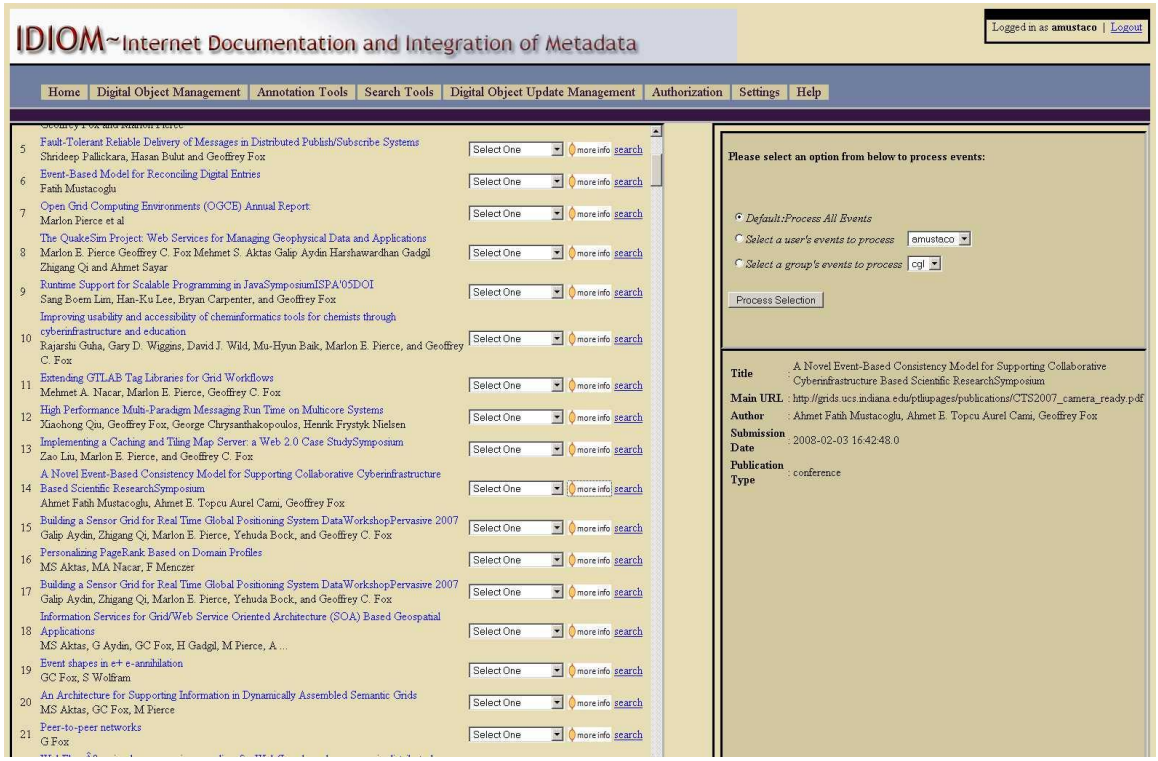

Figure 5-7: MoreInfo of a DE

Based on the coming events type (minor or major), DE Update Management module forwards the requests to the Event and Dataset Management module. To handle the coming update requests, Update DE Metadata service uses *More Info* service to build DE metadata to send back to the requesting clients in XML format. In the current implementation of the IDIOM system, a user can retrieve an existing DE metadata to edit via DE Update Management service. After the user modified any metadata field of the DE, it generates a minor event including the current modification to DE, and stores it into the user's session so that it can be processed later as explained in Section 5.2.3. Once the logged user creates a dataset, then the associated minor events are sent to the DE Update Management module to be processed as minor events. These minor events are forwarded to Event and Dataset Management module to be executed. Furthermore, the user interface of the IDIOM system to insert a new DE sends a request to DE Update Management module to be handled. It forwards this request as a major event to be processed by Event and Dataset Management module. Implemented user interface for Update DE Metadata services are depicted respectively in Figure 5-8. DE Update Management module propagates the update to each annotation tool right away. Updates are disseminated by unicast communication approach since the integrated annotation tools do not support publish/subscribe paradigm by using Communication Manager described in Section 5.3.2.

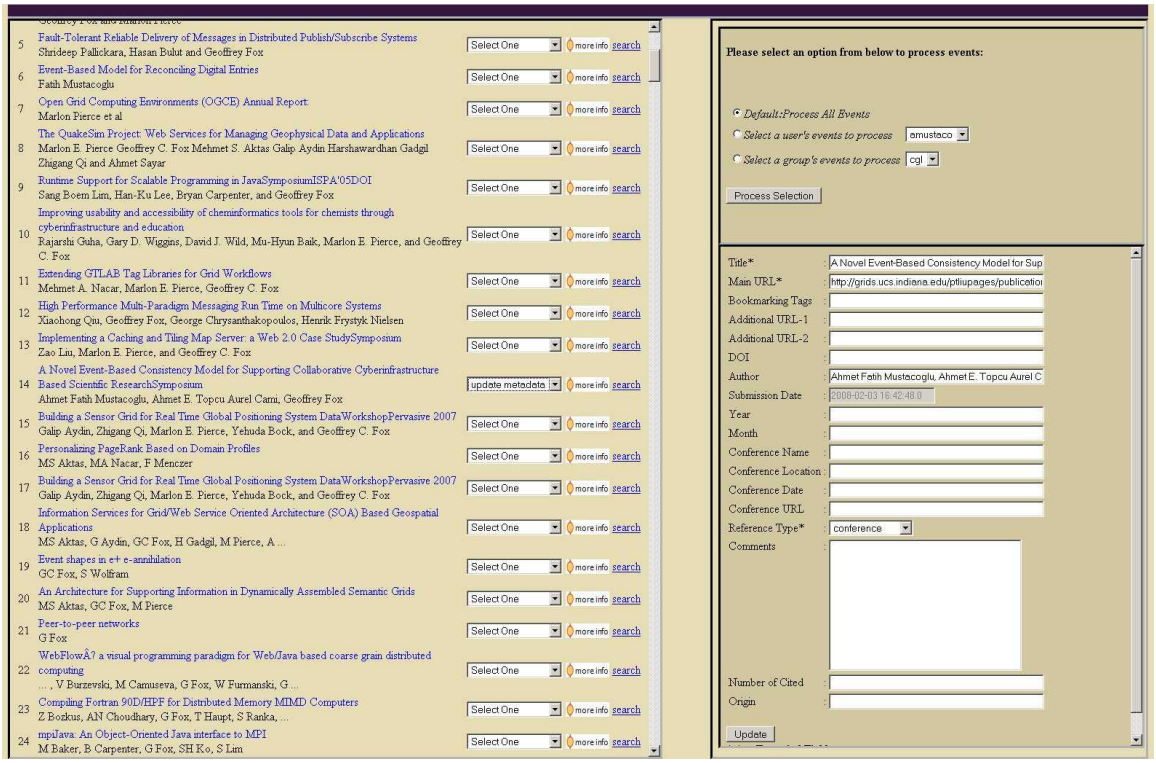

Figure 5-8: Update Metadata of a DE

5.2.5 Search Tools

This module provides services and interfaces to the web-based search tools including Google Scholar, Google Scholar Advanced, and Windows Live Academic. It also provide services for local folder search and integrates the PubsOnline software - "an open source tool for management and presentation of databases of citations via the Web" [164] - into the IDIOM system and providing an interface for searching the logical folders of IDIOM system database. This module is implemented by another PhD student working on this project.

5.2.6 Authentication and Authorization

This module supports IDIOM systems authentication and authorization mechanism to resources including DE and folder access rights structure, super and group role definitions. This module is implemented by another PhD student working on this project.

5.2.7 Other

User Registration, Username and Password Recovery, User's Profile Management, and DE Metadata View Options modules exist in the other modules of the system architecture. These modules are responsible for providing users with services to register with the system, retrieve their forgotten username, reset their forgotten password, manage their profile such as name, email, password etc., and define the view options of digital entities to view or hide specific metadata fields of them.

5.2.8 Timestamp Generator

Timestamp Generator module is responsible for producing uniqueue timestamp values for the requesting processes. In order to impose an order on events, each event has to be time-stamped before it is generated and stored in the session or the MySQL system database. Since, events are processed by Event Processing Engine by their ordered timestamps as explained in Section 3.7.3.1.2. Timestamp values are also used by the consistency mechanism to maintain consistency by imposing an order on updates. To assign a unique timestamp value, Timestamp Generator interacts with Network Time Protocol (NTP) –based time service [165]. This service provides synchronized timestamp

values by synchronizing the distributed machine clocks with atomic time servers available across the universe.

5.2.9 Data Manager

Data Manager is responsible for executing the coming requests on data items. It implements the Data Manager abstract layer as explained in Section 3.7.5. Data Manager uses JDBC connection to connect to MySQL system database.

5.3 Consistency Framework

The Consistency Framework implements optimistic replication approach to ensure eventual consistency between replicas. It propagates and collects the updates by utilizing pull and push based approaches. It is an umbrealla name for: (1) DE Update Management sub-module of the Digital Entity Management Module implementing the push-based approach; (2) Communication Manager; and (3) Annotation Tools Update Manager implementing the pull-based approach.

In the push based approach as explained earlier in Section 5.2.4, DE Update Management module is responsible for disseminating the updates immediately once they occur to the integrated annotation tools via Communication Manager as explained in Section 5.3.2. The updates take place on primary copies located at the central MySQL system database. These primary copies have more metadata field than the DARs located at the annotation tools.

Communication manager is responsible for providing communication between the main system and the integrated annotation tools via their defined gateways explained in detail in Section 5.3.2.

Annotation Tools Update Manager is responsible for checking the integrated annotation tools periodically to obtain the DARs, and compare them to find out the any available updates described in detail in Section 5.3.3. It implements the time-based pull approach to collect the updates from the integrated annotation tools.

5.3.1 Digital Entity Update Management

This module is responsible for handling the updates happened on primary copies of records located at the annotation tools. It pushes the updates to the integrated annotation tools immediately once they occur via unicast methodology. Digital Entity Update Management implements Digital Entity Update Management abstraction layer as explained earlier in Section 3.7.3.2.

5.3.2 Communication Manager

This module provides an interface to the annotation tools: Delicious, CiteULike, and Connotea. It allows a user: (1) to upload DEs data and metadata to one of these annotation websites; (2) to download DEs data and metadata from one of the annotation websites into one of the logical folders of IDIOM system database; (3) to transfer DEs data and metadata between these annotation websites. Communication Manager Module implements the Communication Manager abstract layer as explained in Section 4.4.2.2.

To upload data and metadata from a user's specified logical folder to the specified annotation tool, Communication Manager Module uses the defined gateway abstract layer explained in Section 4.4.2.2.1 for the desired annotation tool. Communication Manager builds the DE metadata from its events and uploads it to the annotation tool.

To download data and metadata, first Communication Manager Module gets records from the specified annotation tool via its gateway. Second, it parses the coming XML result by using JDOM and XPATH technologies. Third, it passes the coming data to DE Management module as explained in Section 5.2.4 in order to be processed and entered into the MySQL database as major event in the case of being a new entry. If they are existing entries, then they are saved into the user's session as minor events.

To transfer data and metadata between annotation tools, Communication Manager Module first retrieves the records from the first annotation tool via its gateway. Second, it parses the coming result that is in XML format by using JDOM and XPATH technologies. Third, it uploads data and metadata to the second annotation website via its gateway.

5.3.3 Annotation Tools Update Manager

Annotation Tools Update Manager module is responsible for implementing a mechanism to deal with the consistency maintenance of DARs located at several annotation tools by using time-based pull approach to collect updates. This module implements the Annotation Tools Update Manager abstraction layer as explained in Section 4.4.2.3.

In the pull based approach, Annotation Tools Update Manager utilizes Java Threads running in the background for each integrated annotation tool. It basically collects records from the annotation tools to find out updates and passes them to DE Management module to be handled. Process of collecting and applying updates requires several steps. First, these threads wakes up periodically to check updates as specified in the properties file of the IDIOM prototype system and communicate with their regarding

annotation tool to get records from there. Second, they gets records from the IDIOM system's database and compares the coming records the ones coming from the database to find out that whether any update or new entry exists. If there is any update, then they are put into a shared hashtable by all threads. If there are two or more updates for the same record, then the updates are processed based on the concurrent access policy as defined in Section 4.4.1 and are put into the shared hashtable where all the updates are collected. Third, this module passes the updates to DE Management Module to be pushed to the integrated annotation tools as explained in 5.2.4 so that primary copies of DEs can be updated and the updates are disseminated to the annotation tools to reflect the changes over there. Update Propagation is explained in detail in Section 5.3.4.

Finally, in our implementation of Consistency Framework Management module of IDIOM system, we have used various technologies. Summary of the technologies are represented in Table 5-1.

Table 5-1: Summary of Technologies

API	Purpose
JDOM	For parsing XML documents
Jakarta Commons HTTP Client version 3.0.1	For handling HTTP communication
XPATH	For querying an XML document object
JTidy	For parsing HTML documents
Apache Axis version 1.2	For creating Java Web Services
JAVA	For implementing the framework

5.3.4 Update Propagation

In distributed systems, there are two approaches for update propagation: Pull approach and Push Approach. In Pull approach, a server or client ask another server to send any updates that it may have, whereas in Push based approach, updates are propagated to other replica server without their requests [110]. In our prototype implementation, we utilized push and time-based pull methodology for update propagation and unicast technique for dissemination of updates to integrated annotation tools. Based on this methodology, whenever an update occur on IDIOM system, the primary-copy immediately reflects the changes to the replica copies located at the integrated annotation tools in order to keep them up-to-date with the recent change. Updates can be distributed in either unicast or multicast communication methodology [110]. In unicast update propagation methodology, the primary-copy server sends updates to replica holders separately, while in multicast update propagation, it send the updates by using an underlying multicast utility that handles sending the updates to replica holders. For dissemination of updates, we used unicast communication methodology due to lack of support for publish/subscribe mechanism of annotation tools.

CHAPTER 6

Prototype Evaluation and Discussions

In this chapter we performed extensive series of measurements to evaluate the prototype implementation of the proposed architecture and investigate its practical usefulness in real life applications. In this chapter, the following research questions are being answered:

- What is the baseline performance of the Consistency Framework implementation in terms of the base upload and download operations? (Section 6.2 answers this question.)
- What is the optimum number of clients that can be concurrently supported by the proposed system? (Section 6.3 answers this question.)

- How well does the system perform when the message rate per second is increased for MoreInfo standart operation with DB access? (Section 6.3 answers this question.)
- How well does the system perform when the message rate per second is increased for MoreInfo standart operation with memory utilization? (Section 6.3 answers this question.)
- How well does the system perform when the message rate per second is increased for Update DE standart operation? (Section 6.3 answers this question.)

6.1 Testing Environment

We tested our Event-based Infrastructure and Consistency Framework implementation by using gf12-15 nodes and gf16 node of clusters located at Community Grids Laboratory at Indiana University. We have run our client programs on gf12-gf15, we have deployed our service-based Event-based Infrastructure and Consistency Framework system on gf16, and we have installed our database on gf16. Summary of these machine configurations are given in Table 6-1, and Table 6-2 respectively.

Table 6-1: Summary of Cluster Nodes – (gf12-15).ucs.indiana.edu

Cluster Nodes gf12-15.ucs.indiana.edu	
Processor	Intel® Xeon™ CPU (E5345 2.33GHz)
RAM	8 GB (each node)
OS	GNU/Linux (kernel release 2.6.9-5.ELsmp

Table 6-2: Summary of Cluster Node - gf16.ucs.indiana.edu

Cluster Node gf16.ucs.indiana.edu	
Processor	Intel® Xeon™ CPU (E5345 2.33GHz)
RAM	8 GB Total
OS	GNU/Linux (kernel release 2.6.9-5.ELsmp

In our general experiments methodology, we have used single-threaded and multi-threaded client programs. Our Event-based Infrastructure and Consistency Framework is also a multi-threaded service-enabled system running on cluster node gf16.ucs.indiana.edu. We have sent various requests from the client programs to our proposed system implementation to test the performance, and the scalability of our proposed system.

We have implemented our service-enabled Event-based Infrastructure and Consistency Framework in Java Language, using Java 2 Standard Edition compiler with version 1.5.0_12. In our experiments with the prototype implementation, we used Apache Tomcat Server with version 5.0.28 and Apache Axis technology with version 1.2 as a container. We set the maximum heap size of Java Virtual Machine (JVM) to1024MB by using the option –Xmx1024m. In our experiments, we also increased the maximum number of threads from default value to 1000 in Apache Tomcat Server to be able to test the system behavior for the huge numbers of concurrent clients.

6.2 System Responsiveness Experiments

Our main goal in doing this experiment is to measure the baseline performance of our Event-based Infrastructure and Consistency Framework implementation. We have tested the performance of our proposed system by measuring the times necessary to download a record from an annotation tool into a repository, and to upload a new record from a repository to an annotation tool (forms a DAR). Furthermore, we have investigated latency values for More Info functionality with DB access and memory utilization, and Update DE functionality. The performance evaluation is done when there is no additional traffic in the system. The primary interest for doing system responsiveness experiment was to investigate the optimum performance of the system for download, upload, more info and update digital entity primary operations for the proposed system. The client programs were running on a cluster nodes gf12-gf15, while service-enabled Event-based Infrastructure and Consistency system was running on a cluster node gf16.

In this experiment, we were exploring the performance of our methodology for download, upload, more info and update digital entity operations of the proposed system. We have conducted the following test cases: a) A single client sends a request to download a DAR from an annotation tool as a major event required to access to the DB; b) A single client sends a request to make a new DAR required to access to an annotation tool; c) A single client sends a request to get a more info on a digital entity from a repository required to access to the DB; d) A single client sends a request to get a more info on a digital entity from the cache required to access to the memory; and e) A single client sends a request to update a digital entity existed in a repository.

In our each testing case, the clients send 400 sequential requests for download, upload, more info and update digital entity standard operations. We recorded the average execution time, and this experiment was repeated 5 times. Figure 6-1 shows the design of these experiments.

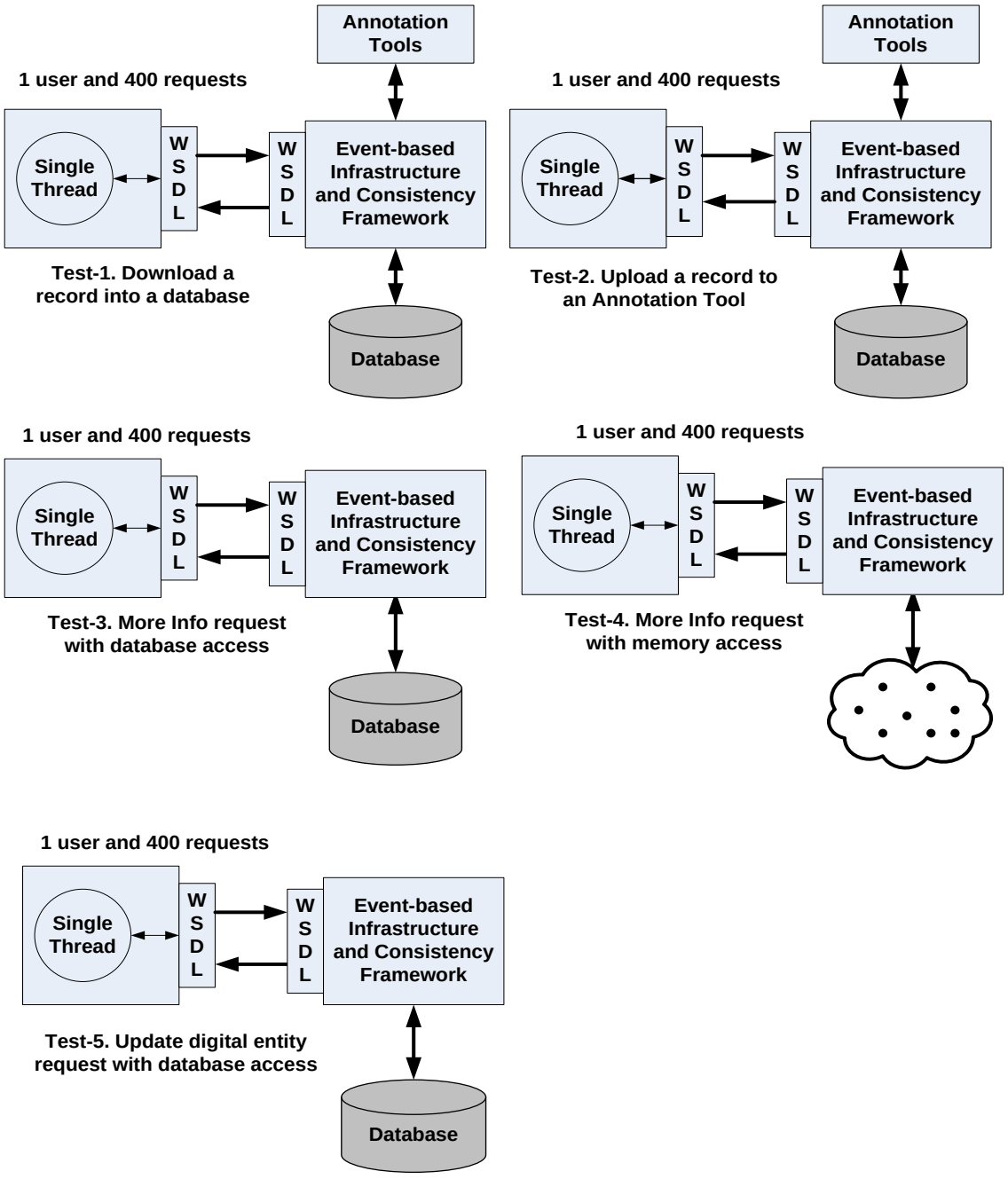

Figure 6-1: Testing Cases for System Responsiveness Experiment

6.2.1 System Responsiveness Experiment Results

We conduct experiments where we investigate the base performance of the proposed system. Depicted in Figure 6-2, Figure 6-3, Figure 6-4, Figure 6-5, Figure 6-6 and listed in Table 6-3, Table 6-4, Table 6-5, Table 6-6, Table 6-7 represent basic proposed system responsiveness result. In this experiment we first recorded execution times for: a) calling the download service to measure the processing time of our implemented service; b) calling the upload service to measure the processing time of our implemented service. Next, we recorded round trip times for: a) calling the More Info service with database access to measure the latency of our implemented service; b) calling More Info service with memory utilization to measure the latency of our implemented service; c) calling Update DE service to measure the latency of our implemented service. Downloading a new entry requires to store this entry as a major event in the database and it is one of the major services provided by the proposed Event-based Infrastructure and Consistency Framework. Furthermore, the proposed Event-based Infrastructure and Consistency Framework propagates the updates via push mechanism by using upload service of the system in order to maintain consistency. This experiment shows the necessary time requirements for these major services to download or to upload a digital entity between the database and annotation tools (replicas).

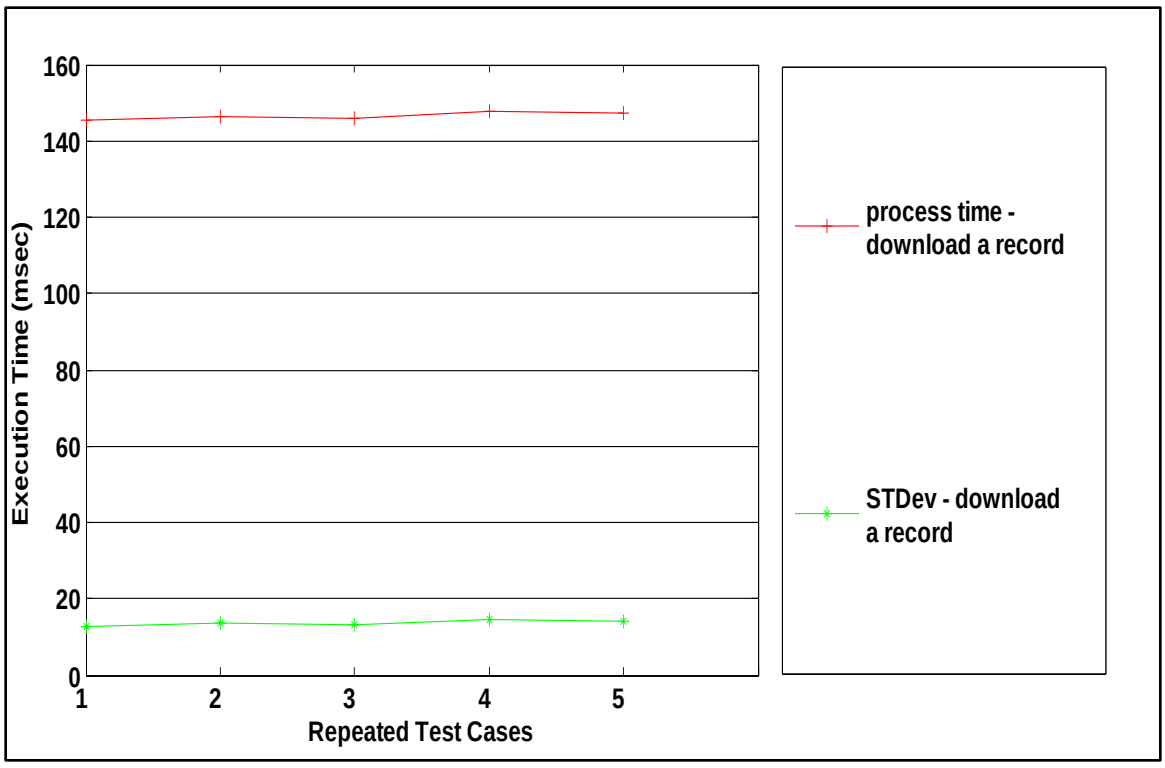

Figure 6-2: Download a record

Table 6-3: Statistics of the experiment depicted in Figure 6-2

Repeated Test Cases	1	2	3	4	5
Process time (msec)	145.44	146.49	145.72	147.77	147.37
STDev	12.74	13.64	13.09	14.54	13.94

Average of 141ms of the process time is coming from the annotation tool. When we do a request to retrieve data from an annotation tool, the annotation tools replies us in average of 141 ms. The biggest part of the processing time is caused by the annotation tools, and the average processing time decreases to 4 ms.

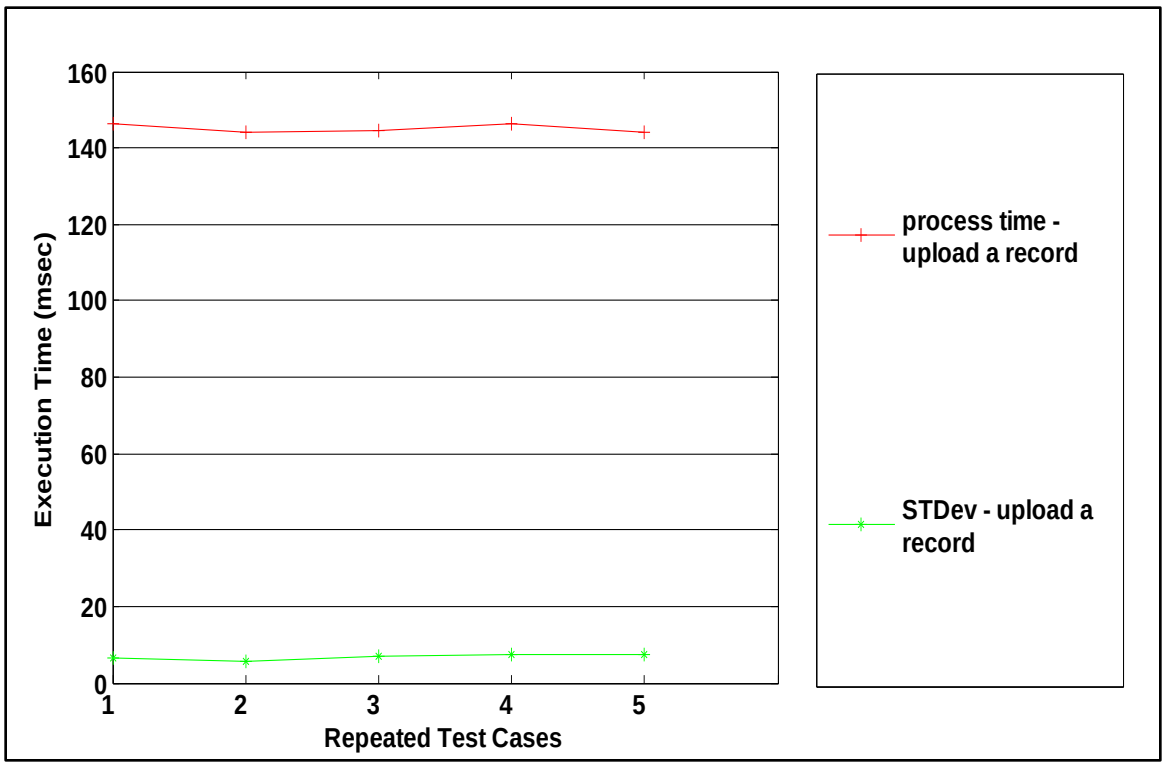

Figure 6-3: Upload a record

Table 6-4: Statistics of the experiment depicted in Figure 6-3

Repeated Test Cases	1	2	3	4	5
Process time (msec)	146.24	144.23	144.75	146.33	144.3
STDev	6.61	5.52	7.11	7.6	7.24

Average of 144 ms of the process time is coming from the annotation tool. When we do a request to retrieve data from an annotation tool, the annotation tools replies us in average of 144 ms. The biggest part of the processing time is caused by the annotation tools, and the average processing time decreases to 2 ms.

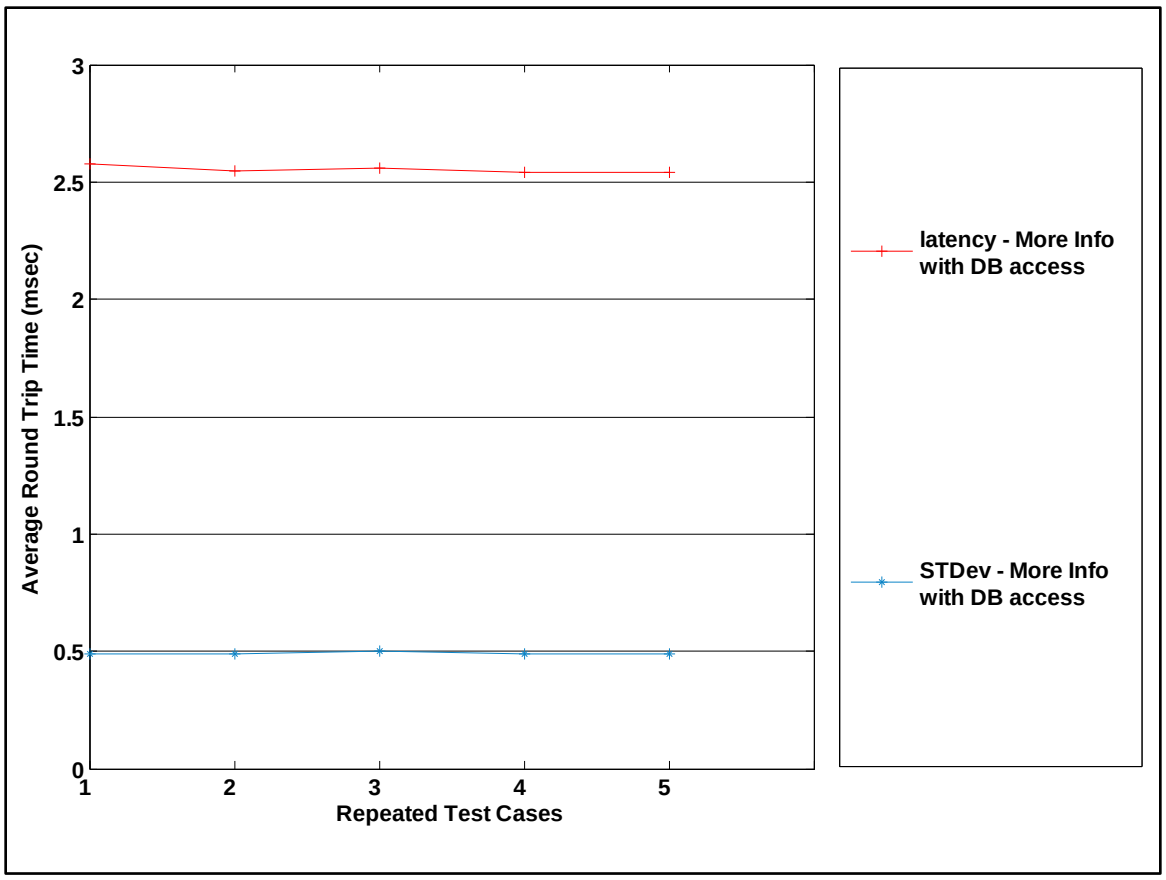

Figure 6-4: Latency and STDev values for More Info standard operation with database access

Table 6-5: Statistics of the experiment depicted in Figure 6-4

Repeated Test Cases	1	2	3	4	5
Latency (msec)	2.58	2.55	2.56	2.54	2.54
STDev	0.49	0.49	0.50	0.49	0.49

Average round trip time for a major More Info operation on a DE to retrieve its metadata from a database can result in about 2.5 ms. And this shows that our service side processing does not require much time to process read operation on a DE.

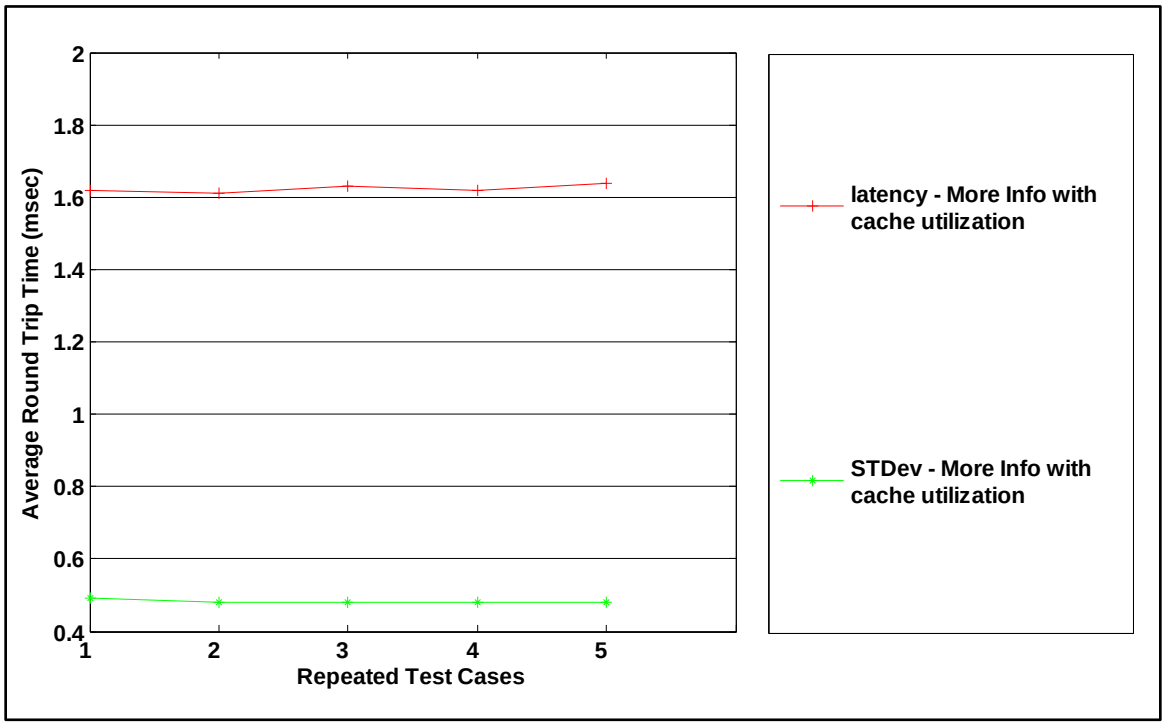

Figure 6-5: Latency and STDev values for More Info standard operation with memory utilization

Table 6-6: Statistics of the experiment depicted in Figure 6-5

Repeated Test Cases	1	2	3	4	5
Latency (msec)	1.62	1.61	1.63	1.62	1.64
STDev	0.49	0.48	0.48	0.48	0.48

Average round trip time for a major More Info operation on a DE to retrieve its metadata from memory can result in about 1.6 ms. And this shows that our service side processing does not require much time to process read operation on a DE.

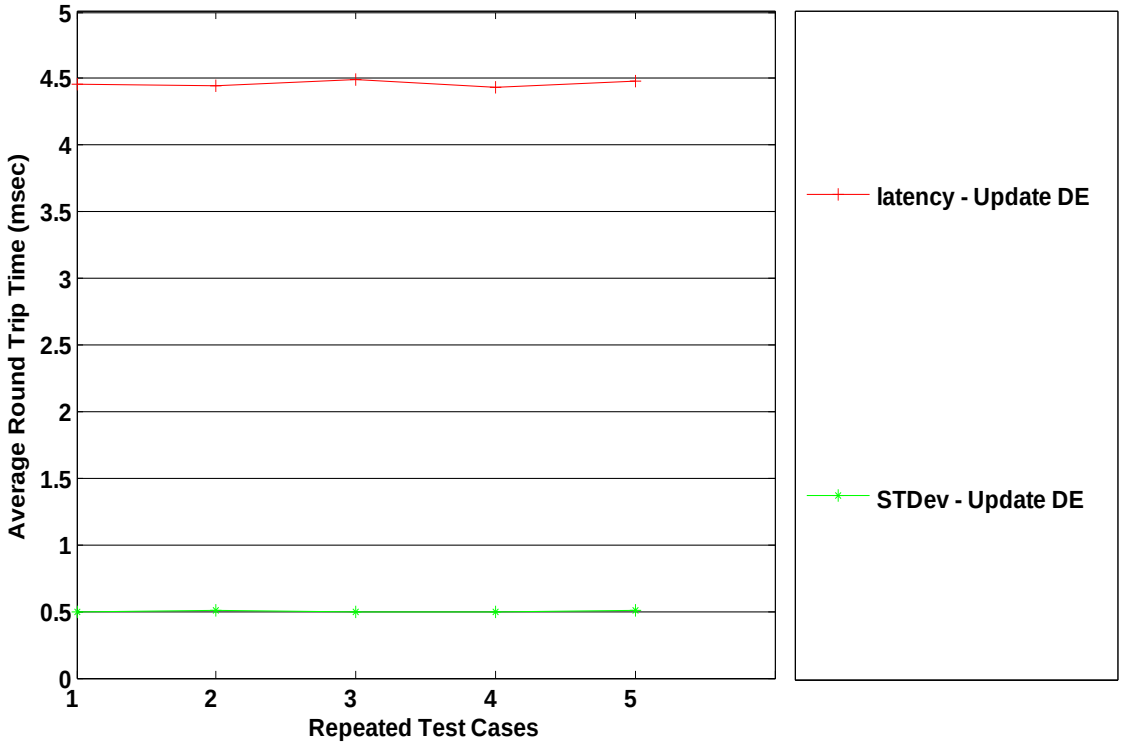

Figure 6-6: Latency and STDev values for Update DE standard operation

Table 6-7: Statistics of the experiment depicted in Figure 6-6

Repeated Test Cases	1	2	3	4	5
Latency (msec)	4.46	4.45	4.49	4.43	4.48
STDev	0.49	0.51	0.50	0.49	0.51

Average round trip time for a major Update operation on a DE to modify its content can result in about 4.4 ms. And this shows that our service side processing does not require much time to process write operation on a DE. When there is a multiple write operation on a same DE, database will lock access to its resources.

6.3 Scalability Experiment

The primary interest in doing this experiment was to investigate the scalability of Event-based Infrastructure and Consistency Framework implementation. We conducted three testing cases and tried to answer the following research questions: a) how well does the system performs when the message rate per second is increased for More Info standard operation request on a DE with DB access?; b) how well does the system performs when the message rate per second is increased for More Info standard operation request on a DE with memory utilization?; b) how well does the system performs when the message rate per second is increased for Update DE standard operation request?

In first experiment, our main goal is to identify the number of concurrent requests requiring DB access that can be handled by the proposed system when message rate per second are increased in the Event-based Infrastructure and Consistency Framework. We have completed this test case by increasing the message rate/sec until the response time degrades. In this testing case, we recorded round trip time at each MoreInfo request on a DE with DB access. In the second testing case, we have applied the same technique as previous experiment except that each request is responded by using memory utilization. In the third experiment, we have investigated the concurrent requests for an Update DE main operation that can be serviced by the Event-based Infrastructure and Consistency Framework while message rate per second are increased. The designs of these testing cases are depicted in Figure 6-7.

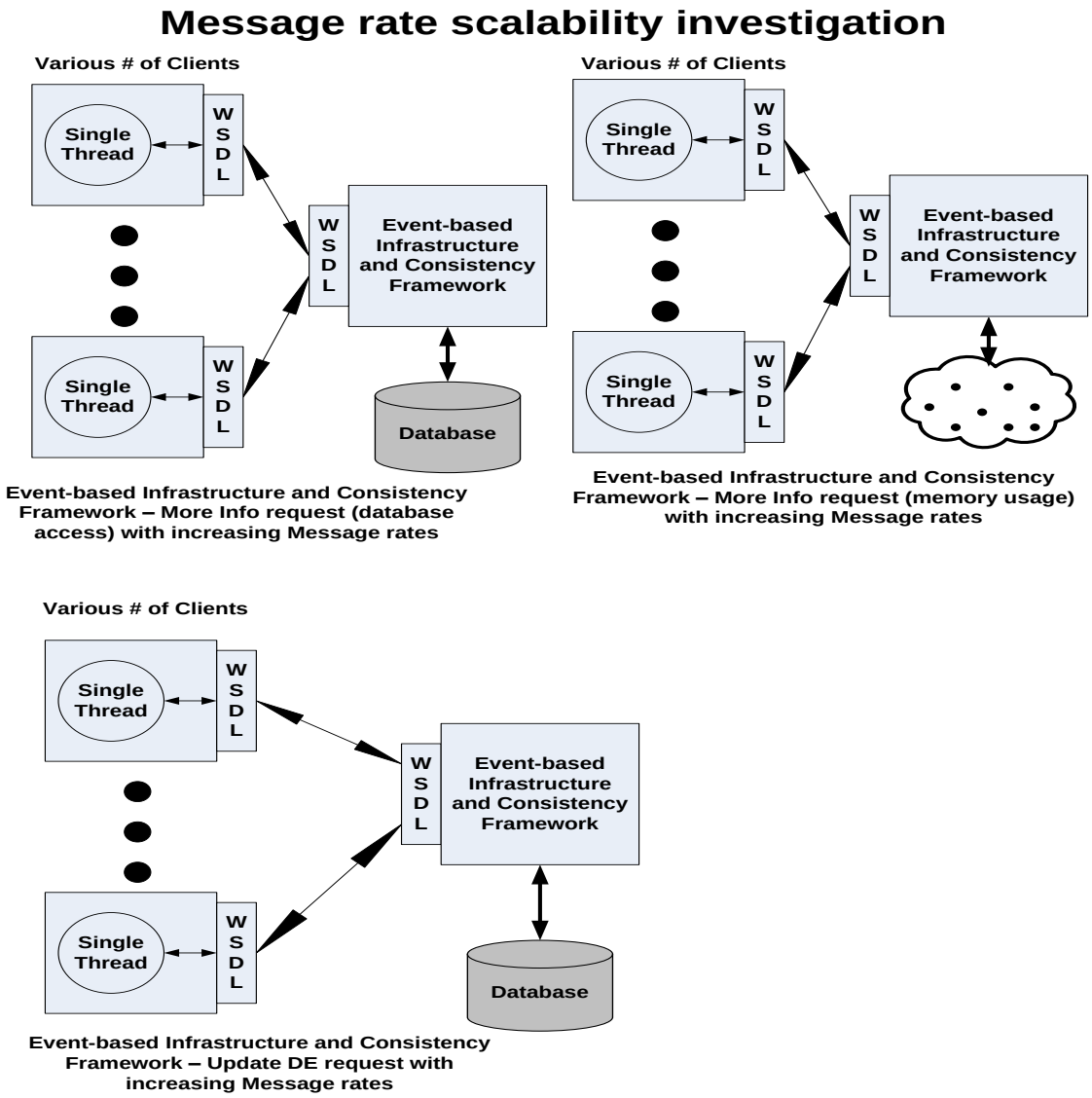

Figure 6-7: Testing cases of scalability experiment for More Info and Update DE requests

6.3.1 Scalability Experiment Results

Based on the results depicted in Figure 6-8 and listed in Table 6-8, we determined that concurrent inquiry requests may be well responded by Event-based Infrastructure and Consistency Framework without any error. According to the experiment result, we identified that Event-Based Infrastructure and Consistency Framework major operations

performed well for the increased message rate. However, after a certain number of messages per second, performance starts to degrade due to high message rate. We observe that after around 1060 inquiry messages per second for MoreInfo with DB access, after around 2068 inquiry messages per second for MoreInfo with memory utilization, after around 533 inquiry messages per second for Update DE, the system performance degrades due to high message rate. This threshold is mainly due to Apache Tomcat (thread scheduling and context switches) as explained in Section Figure 6-11. Experiment results are depicted in Figure 6-8, Figure 6-9, Figure 6-10 and listed in Table 6-8, Table 6-9, Table 6-11.

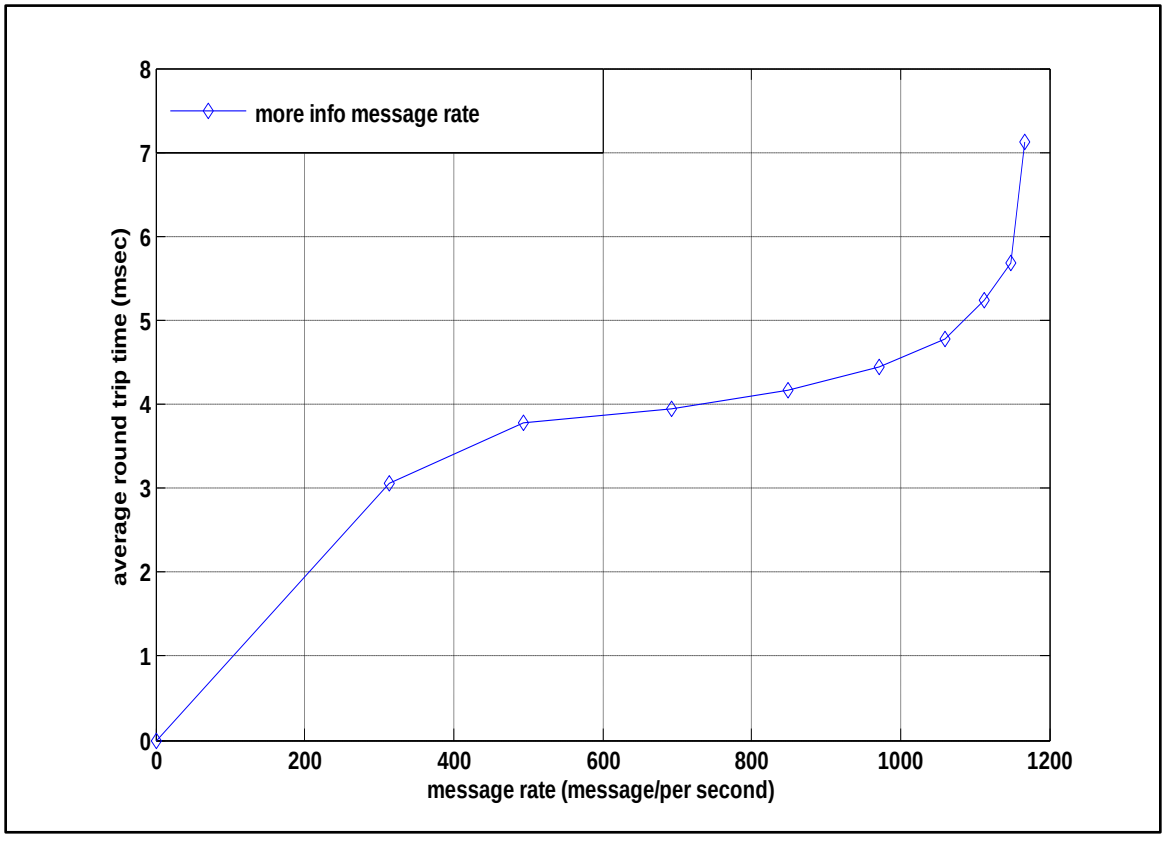

Figure 6-8: More Info message rate with DB access

Table 6-8: Statistics of the experiment results depicted in Figure 6-8. Time units are in milliseconds

Event-based Infrastructure and Consistency Framework – Increased Message Rate	
Messages/second	Average Timing (msec)
0	0
314	3.06
494	3.78
692	3.95
849	4.16
972	4.44
1060	4.78
1113	5.24
1148	5.69
1167	7.12

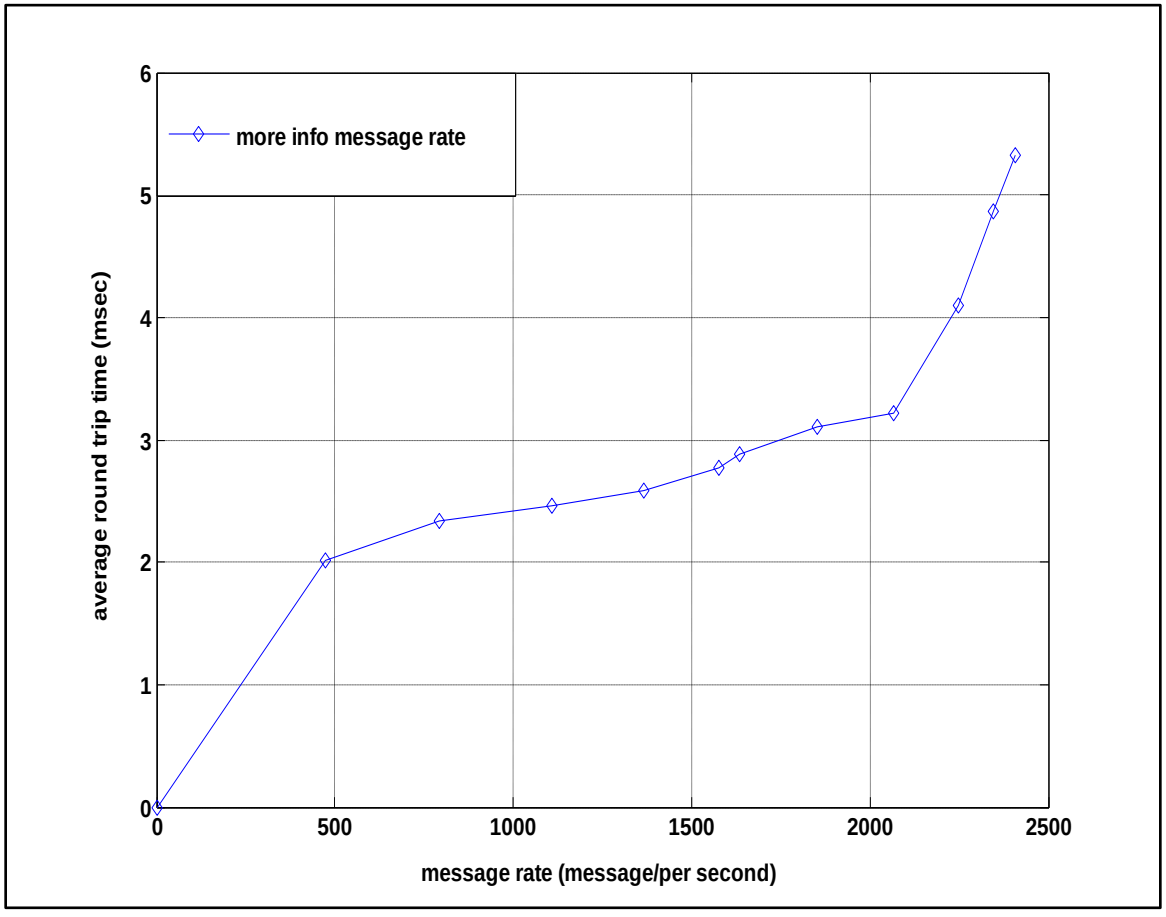

Figure 6-9: More Info message rate with memory utilization

Table 6-9: Statistics of the experiment results depicted in Figure 6-9. Time units are in milliseconds

Event-based Infrastructure and Consistency Framework – Increased Message Rate	
Messages/second	Average Timing (msec)
0	0
474	2.01
793	2.34
1109	2.46
1367	2.59
1578	2.77
1636	2.88
1853	3.11
2068	3.22
2247	4.10
2346	4.86
2408	5.32

Overhead calculations for improving the performance for MoreInfo operation with DB and memory utilization will be the time necessary to calculate the latest version of a DE after applying a coming update on top of it and storing back the latest version into the database. Statistics of the experiment is listed in Table 6-10.

Table 6-10: Statistics of the overhead calculations for database and memory utilization to improve the performance. Time units are in milliseconds

Overhead Time (Database)	STDev for DB	Overhead Time (Memory)	STDev for Memory
6.82	0.75	0.96	0.35

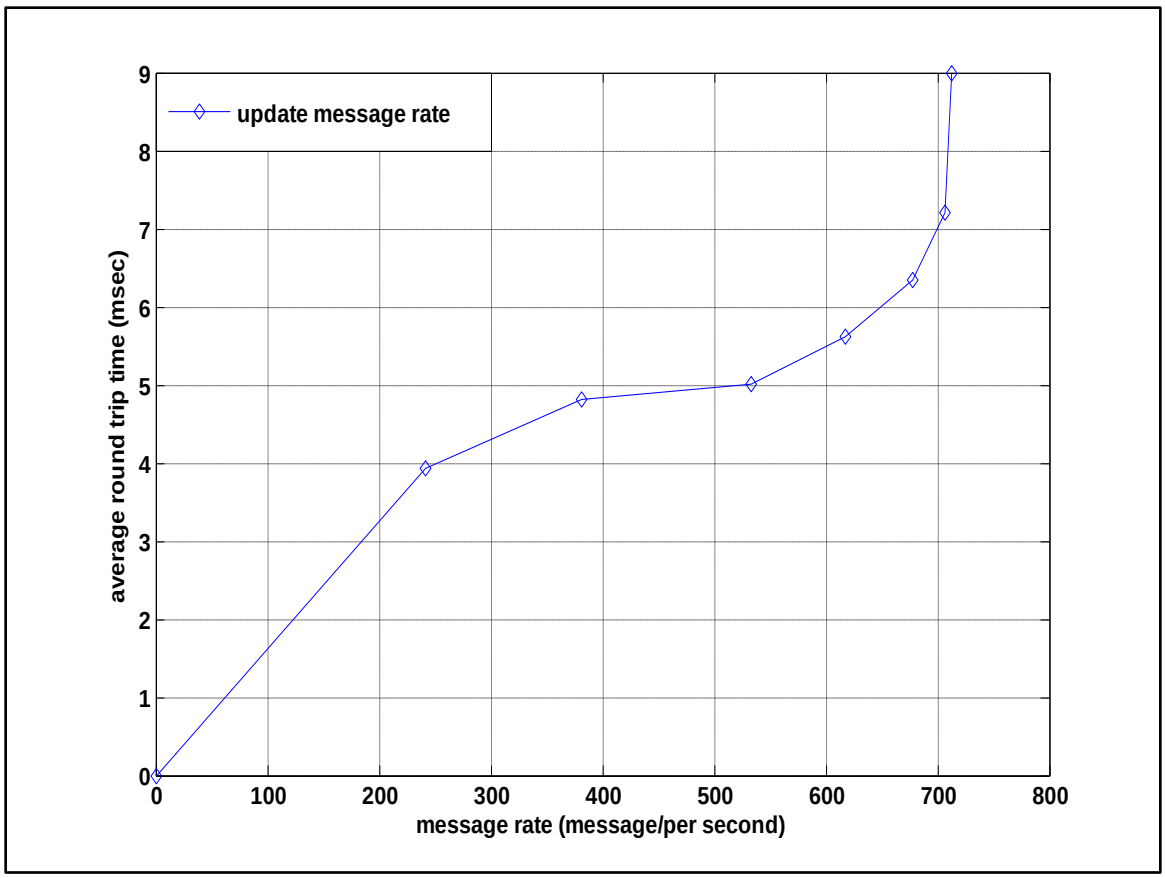

Figure 6-10: Update DE message rate

Table 6-11: Statistics of the experiment results depicted in Figure 6-10. Time units are in milliseconds

Event-based Infrastructure and Consistency Framework – Increased Message Rate	
Messages/second	Average Timing (msec)
0	0
241	3.93
381	4.82
533	5.01
617	5.62
678	6.34
707	7.20
713	8.99

6.3.1.1 Investigation of the threshold value in scalability graphs

To investigate the reasons of the threshold value, we have investigated the possible causes for the threshold value: (a) Network bandwidth investigation; (b) Limitation on open sockets in Linux; (c) Tomcat limitations such as thread scheduling and context switches.

6.3.1.1.1 Network Bandwidth Investigation

In this section, we have measured a message size and calculated the total network need to see whether this threshold value is due to the network bandwidth or not.

- Message size in empty service method call is 466 bytes.

  Message size in bits 466 bytes * 8 = 3728 bits

  A total network is needed at the threshold value is:

  3,738 bits/message * 3,693 message/sec = 13.8 Mbits/sec

- Message size in More Info request is 879 bytes.

  Message size in bits 879 bytes * 8 bits = 7,032 bits

  A total network is needed at the threshold value is:

  7,032 bits/message * 2,068 message/sec = 14.5 Mbits/sec

- Message size in Update metadata request is 3,700 bytes.

  Message size in bits 3,700 bytes * 8 bits = 29,600 bits

  A total network is needed at the threshold value is:

  29,600 bits/message * 533 message/sec = 15.8 Mbits/sec

Discussion: Our network capability in CGL is 1GBits/sec. In the first case, its value is almost %1 percent of the network capacity. So, this cannot be the reason for this threshold value. In the second case, its value is also almost %1 percent of the network

capacity. So, this cannot be the reason for this threshold value as well. In the third case, its value is also almost %1 percent of the network capacity. So, this cannot be the reason for this threshold value as well.

So, finally we concluded that the network bandwidth cannot be the cause for the threshold value in these figures.

6.3.1.1.2 Limitation on open sockets in Linux

As default, each user has 1024 open socket connections in Linux. We have performed our scalability tests with the increased open sockets from 1024 to 2048, and we have retrieved the similar results that we obtained with the 1024 open socket connections. So, we have concluded that the numbers of allowable open sockets are not the cause for our threshold value in our graphs.

6.3.1.1.3 Apache Tomcat limitations

In this section, we have investigated that the threshold value is occurring due the tomcat limitations. To test whether tomcat causing this threshold value or not, we have implemented an empty service method that has nothing in it with no parameters. We have measured the round trip time while we increase the message rates with this empty service method calls. Figure 6-11 represents our investigation results.

Finally, we have concluded based on the results that we obtained in Figure 6-11 that the reason for the threshold value is due to Apache Tomcat limitations (thread scheduling and context switches to satisfy the coming requests at high message rates) since we are obtaining the same pattern with an empty service call measurements.

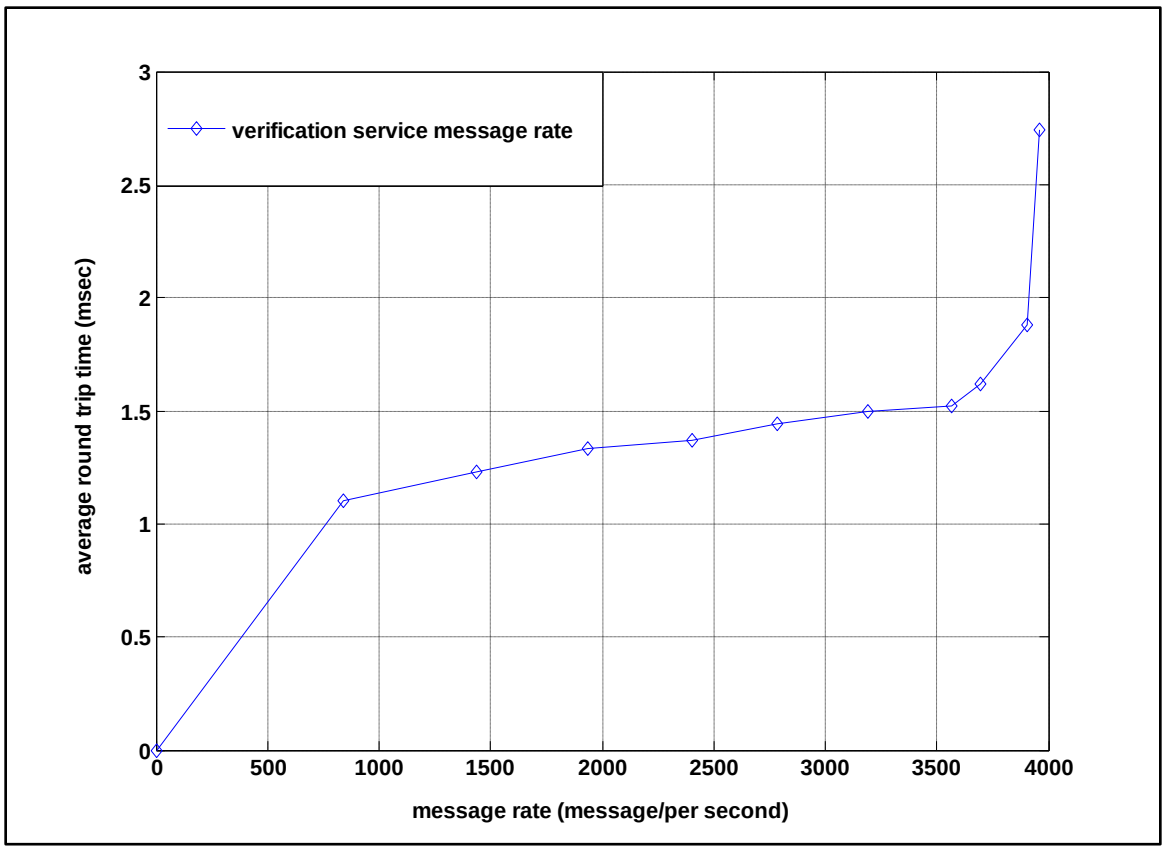

Figure 6-11: Verification Service Message Rate

6.4 Summary

This chapter presented the performance evaluation of our proposed Event-based Infrastructure and Consistency Framework. First, our experiment pointed out the trade-off between the scalability and performance of the proposed system. Based on the experiment results, we discovered some threshold values for the maximum number of simultaneous Update DE, and More Info service operation with DB access and memory utilization that can be performed on the system. For instance, while the number of requests exceeds 1060 simultaneous messages per second for More Info operation with DB access, the system performance starts to decrease. This experiment results also showed that the system is able to scale to increasing message sizes and performs well.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Thesis summary

This thesis studied Event-based Infrastructure and a Consistency Framework for reconciling distributed annotation records located at various annotation tools. We have identified our motivation, and research problems in CHAPTER 1. We discussed the related work and survey of technologies in CHAPTER 2. Having identified our motivations, and reviewed the relevant research works, we proposed our architectural design for an Event-based Infrastructure and Consistency Framework for Distributed Annotation Records. We introduced the Event-based Infrastructure in CHAPTER 3. We presented the Consistency Framework for Distributed Annotation Records (CFDAR) in

CHAPTER 4. We explained our proposed research's implementation details in CHAPTER 5. We introduced prototype evaluation and discussions in CHAPTER 6.

Social bookmarking services strikingly changed how people find and refind information on the internet. Furthermore, it demonstrated the power of online collaboration and there are several features of social bookmarking services that support collaboration. First, users of these systems store data and metadata in a shared place instead of storing them in a private local storage such as a personal computer's hard drive. This allows anyone to access other users' records resulting in users to discover new sources of information. Second, social bookmarking services leverage the use of collaborative tagging of resources. Users can tag their content on these social bookmarking tools, organiza their resources by tags, and search all resources existed on social bookmarking services by tags to find related or interested resources. So, collaborative tagging provides users with ability to organize their resources in a flexible way and to access related research materials easily.

There has been an enormous growth in social bookmarking applications and the most obvious and famous example is del.icio.us. Social bookmarking tools offer great services for publicly accessible web resources. Today, there are various types of social bookmarking services focused on different areas such as Social Networking Tools (MySpace, LinkedIn), Social Bookmarking Tools (del.icio.us), Video Sharing and annotation (YouTube), and annotation and sharing of scholarly publications (CiteULike, Connotea, Bibsonomy).

Despite the huge number of web-based annotation tools that provide services for storage and collaboration of resources over the internet, these tools have limitations. First

of all, their metadata support for the stored documents varies and the metadata fields provided by these tools are not enough to represent the whole content of a scientific document causing storing same scholarly publications in several annotation tools. Second, these tools are lack of support for communication with each other. Third, they also suffer lack of services to upload data from a repository, extract and import data into a repository. Finally, they do not provide timestamp information for the updated records causing inconsistencies once documents get updated.

Our thesis research focused on reconciling distributed annotation records stored at the social bookmarking services, which enable annotation and sharing of scientific content, with additional metadata support and capabilities. We have reviewed events sytems in Section 2.2 and overviewed consistency models for distributed systems in Section 2.3. We have investigated an Event-based Infrastructure and a consistency mechanism by adopting existing consistency models for distributed systems to our research.

A promising approach to address the above issues is the event-based paradigm and providing a consistency mechanism around it to keep replica records stored at several annotation tools consistent with each other. The components of an event-based system cooperate by sending and receiving events, a particular form of messages. Our event-based infrastructure is the key concept to our research. Documents, metadata, and modifications to them are represented as events in our research. Events allow us to keep track of changes to documents and metadata. It also provides users with ability to rollback in a flexible fashion to change the state of a digital entity referred as DE in the previous chapters.

Event-based infrastructure benefits from representing documents as events. Since, we never loose a version of a document. It provides flexibility of having different versions of a document any time and enables going back and forward among the versions of a document. Furthermore, event-based approach allows us to handle various types of metadata coming from several sources as events such as annotation tools (Connotea, Bibsonomy, and CiteULike), academic search tools (Google Scholar, Windows Live Academic).

Proposed Event-based Infrastructure provides a comprehensive metadata support to represent the complete metadata of a document in its system. Moreover, it supports collaborative tagging of documents that brings many advantages such as organizing data and metadata in a non complex way, grouping or accessing interested documents by clicking on a related tag or a user etc.

Event-based Infrastructure also has advantages of having timestamp values for each action made on a document. Events can be ordered and executed based on their timestamp values. Furthermore, concurrent updates on a shared document can also be handled based on our concurrent update policy as explained in Section 4.4.1.

Our Consistency Framework for Distributed Annotation Records (CFDAR) also benefits from having an Event-based Infrastructure by adopting optimistic replication approach to ensure eventual consistency. It utilizes push-based approach to propagate updates and time-based pull approach to retrieve the updates from the annotation tools. CFDAR propogates updates by using Digital Entity Update Management module to all replicas based on our push-based update propagation approach once updates occurred on a primary copy of documents. Propagation of updates is done via unicast communication

due to missing support of publish/subscribe mechanism in the integrated annotation tools. However, we have adopted a time-based approach for pulling updates from annotation tools periodically to apply primary copies and rest of the replicas to make them consistent with each other.

Web Service support in our Event-based Infrastructure is also another key feature to allow different client running on a different platforms to interoperate with each other. Our services can be accessed via SOAP calls to access documents from any client that has internet access.

Our Event-based Infrastructure and Consistency Framework scales very well. We have performed several evaluations to measure scalability, and performance of our proposed Event-based Infrastructure and Consistency Framework in CHAPTER 6.

Our Event-based Infrastructure and Consistency Framework is a very flexible system. It unifies and federated major annotation tools and it easily allows adding new annotation or academic search tools into the system. The only necessary action is to implanting a suitable gateway as explained in detail in Section 4.4.2.2.1.

7.2 Answering the research questions

In this section, we will answer our research questions raised in Section 1.2.

Can we implement an infrastructure that handles data and metadata coming from various sources in Service Oriented Architecture? Can this infrastructure unify and federate various existing online annotation tools for publications, which stores replicas of the same documents, and use their services? What is the efficient and flexible data model for such framework?

The answer to this question is “yes” with some limitations. We introduced an Event-based Infrastructure that can deal with various data and metadata coming from different sources. Our Event-based Infrastructure also supports Web Service technology to leverage interoperability among different clients. CHAPTER 3 overviewed the event-based architecture, and CHAPTER 5 explained the prototype implementation of the architecture. Event-based Infrastructure approach provides efficient and flexible data model for handling data and metadata coming from various sources. Furthermore, it builds the necessary layers to unify and federate the various annotation tools into the proposed research framework. Unification and federation of annotation tools require a common access interface to access the annotation tools and a common schema to handle the metadata coming from the annotation tools. A gateway structure also needs to be deployed into the system for each annotation tool that needs to be unified in the system. Furthermore, schema file need to be updated for each annotation tool that need to be federated by the proposed system.

How can we support a flexible architecture that allows user to easily track documents?

Event-based Infrastructure is the promising approach to easily track changes to documents and metadata. In this approach, every action made on a document is kept with a timestamp value. This allows us to keep a history of each document and their modifications ordered by time. Timing of the events is provided by the timestamp generator that uses Network Time Protocol (NTP) implementation from CGL, and system relies on this service for getting timestamp values for the events. This NTP-based service provides an accuracy of 1-30 ms range. There could be cases where 1 or 2 millisecond

can be important. Event-based Infrastructure and its details are given in detail in CHAPTER 3.

How can we provide a consistency mechanism between the online replicated documents stored at annotation tools for scholarly publications and documents located on a central server?

Consistency maintenance is all about keeping all copies of data that may possibly be distributed to different locations to be the same. After reviewing consistency models, protocols and update propogations for distributed systems in Section 2.3, we have designed our Consistency Framework for Distributed Annotation Records (CFDAR) to maintain consistency for distributed annotation records (DARs) located at various annotation tools. CFDAR is a promising approach to maintain consistency among DARs and their primary copies via pull and push based update propagation models. CFDAR is a client-centric consistency model and it adopts optimistic replication approach and ensures eventual consistency between replicas. Whenever updates occurred on a primary copy of a DAR, they are being propagated immediately to each annotation tool to update replication of same document kept at various annotation tools (push approach). However, in time-interval based approach, we periodically check DARs from each annotation tool for any updates. If there is any, then we are pulling them out and applying them on the primary copy of each DAR, which is stored in a relational database with additional metadata, and propagating them to the integrated annotation tools. Our optimistic replication approach maintains consistency between the replicas. However, there might be some cases where updates might get lost. This could be due to the nature of the annotation tools since they do not support notification systems. So, if multiple updates

occur in the annotation tools before we perform our periodic check, we would not be able to track the earlier updates since the last periodic check. We could only get the last update as the only one update by our periodic check. CHAPTER 4 discussed the proposed CFDAR in detail.

How can we achieve an information management architecture that can provide more metadata support than the current annotation tools do for scholarly publications?

In current annotation tools that hold scientific documents metadata support is very limited to include all metadata about a document. Hence, documents are represented with missing metadata field in those tools. In other words, documents are kept at these annotation tools are not complete, and they are stored in various annotation tools due to their various services and metadata supports. In our Event-based Infrastructure we keep primary copies of each document with additional metadata support in a central repository consistent with all replicas of the primary copies. Supported metadata field of a document as an event is displayed in Figure 5-2.

Can we support communication between annotation tools for scholarly publications?

The answer to this question is "yes" with some limitations. Existing annotation tools for keeping scientific documents on their site are lack of communication with each other. The interoperability among annotation tools can be leveraged by Web Service technology. Our Event-based Infrastructure has interfaces that provide heteregenous clients to access its services via SOAP calls over HTTP. Our system relies on the API of currently federated annotation tools to use them, and if their API is changed in the future, then also their implementations are need to be changed in the system. Furthermore,

proposed system provides services to communicate with the integrated annotation tools. If any of the integrated tools is down, then our system would not be able upload or download metadata from/to them until they are up and running. It is explained in CHAPTER 3.

How can we provide users with ability to access previous versions of an updated document? Can we allow users to retrieve and apply other users' updates for a same document? What is the flexible update model?

Proposed Event-based Infrastructure (EBI) keeps each document when they entered into the system as a major event and the following updates to it as minor events in a central repository as explained in Section 3.1 and depicted in Figure 3-2. So, we never loose a version of a document and this provides users with ability to have histories of each document at any given time. Users can easily search updates for the similar documents easily to retrieve and apply their updates on their existing records. We have an approach to provide a flexible and efficient update model, which allows users to have ability to ignore or apply updates to the selected documents, explained in detail in Section 3.5.

Does event-based approach scales very well?

The answer to this question is "yes". We have performed measurements to evaulate scalability and performance of our Event-based Infrastructure. Details of the scalability results are given in Section 6.3. Furthermore, we have investigated the base performance of our prototype system. Further details of our prototype implementation evaluations can be found in CHAPTER 6.

Can we support services for extracting data and metadata from these annotation tools into a specified repository? Moreover, can we support services for uploading data and metadata from a repository to annotation tools?

The answer to both questions is "yes". Our Event-based Infrastructure and Consistency Framework for Distributed Annotation Records supports services to communicate with the integrated annotation tools. These services provide users with ability to extract and download data and metadata into a specified repository. Furthermore, users can upload data and metadata from a specified repository to the integrated annotation tools. These concepts and their implementations are defined in CHAPTER 3 and Section 5.3.2 respectively.

7.3 Future research

This thesis deploys an Event-based Infrastructure and adopts a consistency technique for distributed systems to maintain consistency among distributed annotation records and their primary copies stored at a central repository. It introduces an Event-based Infrastructure and utilizes optimistic replication approach to ensure eventual consistency between distributed annotation records representing scholarly publications. We plan to expand on this approach to be able to apply other application domains such as video collaboration domain (YouTube etc.). We will further research machine learning techniques to identify typing errors within the documents. An additional area that we intend to research is to migrate from centralized structure to decentralized structure.

References

[1] A. David, B. Ron, and C. Mark, "Information archiving with bookmarks: personal Web space construction and organization," in *Proceedings of the SIGCHI conference on Human factors in computing systems*. Los Angeles, California, United States: ACM Press/Addison-Wesley Publishing Co., 1998.

[2] G. Fox, A. F. Mustacoglu, A. E. Topcu, and A. Cami, "SRG: A Digital Document-Enhanced Service Oriented Research Grid," presented at Information Reuse and Integration IRI-07, Las Vegas, NV, USA, 2007.

[3] A. F. Mustacoglu, A. E. Topcu, A. Cami, and G. Fox, "A Novel Event-Based Consistency Model for Supporting Collaborative Cyberinfrastructure Based Scientific Research," in *Collaborative Technologies and Systems CTS 2007 in Technical Cooperation with The IEEE Computer Society*. Orlando, FL, USA: IEEE Computer Society, 2007.

[4] A. E. Topcu, A. F. Mustacoglu, G. Fox, and A. Cami, "Integration of Collaborative Information Systems in Web 2.0," 2007.

[5] G. C. Fox, M. E. Pierce, A. F. Mustacoglu, and A. E. Topcu, "Web 2.0 for E-Science Environments," 2007.

[6] A. F. Mustacoglu and G. Fox, "Hybrid Consistency Framework for Distributed Annotation Records in a Collaborative Environment " presented at The 2008 International Symposium on Collaborative Technologies and Systems (CTS 2008), Irvine,CA, 2008.

[7] Apache Axis version 1.x. Available from: http://ws.apache.org/axis/

[8] Apache Tomcat with version 5.0.28. Available from: http://tomcat.apache.org

[9] Sun Microsystems. Available from: http://www.sun.com/

[10] T. Hammond, T. Hannay, B. Lund, and J. Scott, "Social Bookmarking Tools (I): A General Review," in *D-Lib Magazine*, vol. 11, 2005.

[11] Blogger web site. Available from: http://www.blogger.com

[12] Wiki - Wikipedia web site. http://en.wikipedia.org/wiki/Wiki

[13] Wikitravel web site. Available from: http://wikitravel.org

[14] MySpace web site. Available from: http://www.myspace.com/

[15] LinkedIn web site. Available from: http://www.linkedin.com/

[16] Delicious web site. Available from: http://del.icio.us

[17] Flickr web site. Available from: http://flickr.com

[18] YouTube web site. Available from: http://www.youtube.com/

[19] Netvibes web site. Available from: http://www.netvibes.com/

[20] YourLiveWire web site. Available from: http://www.yourlivewire.net/

[21] CiteULike web site. Available from: http://www.citeulike.org

[22] Connotea web site. Available from: http://www.connotea.org

[23] B. Lund, T. Hammond, M. Flack, and T. Hannay, "Social Bookmarking Tools (II): A Case Study - Connotea," in *D-Lib Magazine*, vol. 11, 2005.

[24] BibSonomy web site. Available from: http://www.bibsonomy.org

[25] G. Begelman, P. Keller, and F. Smadja, "Automated Tag Clustering: Improving search and exploration in the tag space," in *World Wide Web Conference*. Edinburgh,Scotland, 2006.

[26] C. Cattuto, V. Loreto, and L. Pietronero, "Collaborative Tagging and Semiotic Dynamics," *PNAS*, vol. 104, pp. 1461, 2007.

[27] R. Lambiotte and M. Ausloos, "Collaborative Tagging as a Tripartite Network," in *Computational Science â€“ ICCS 2006*, 2006, pp. 1114-1117.

[28] S. A. Golder and B. A. Huberman, "Usage patterns of collaborative tagging systems," *Journal of Information Science*, vol. 32, pp. 198-208, 2006.

[29] LibraryThing web site. Available from: http://www.librarything.com

[30] 43things web site. Available from: http://www.43things.com

[31] A. Mathes, "Folksonomies: Cooperative classification and communication through shared metadata," 2004.

[32] W. Xian, Z. Lei, and Y. Yong, "Exploring social annotations for the semantic web," in *Proceedings of the 15th international conference on World Wide Web*. Edinburgh, Scotland: ACM, 2006.

[33] R. Sinha, "A cognitive analysis of tagging (or how the lower cognitive cost of tagging makes it popular)." Available from: http://www.rashmisinha.com/archives/05_09/tagging-cognitive.html

[34] P. Merholtz, "Clay Shirky's Viewpoints are Overrated". Available from: http://www.peterme.com/archives/000558.html

[35] C. Shirky, "Ontology is Overrated: Categories, Links, and Tags ". Available from: http://www.shirky.com/writings/ontology_overrated.html

[36] L. Gordon-Murnane, "Social bookmarking, folksonomies, and Web 2.0 tools " *Searcher*, vol. 14, pp. 26-38, 2006.

[37] JSON. Available from: http://www.json.org/

[38] RSS Advisory Board web site. Available from: http://www.rssboard.org/

[39] Nature Publishing Group web site. Available from: http://www.nature.com

[40] A. Hotho, R. Jaschke, C. Schmitz, and G. Stumme, "BibSonomy: A Social Bookmark and Publication Sharing System," presented at 14th Int. Conference on Conceptual Structures, Aalborg, Denmark, 2006.

[41] BibTeX web site. Available from: http://www.bibtex.org/

[42] LaTeX – A document preparation system. Available from: http://www.latex-project.org/

[43] JavaServer Pages Technology. Available from: http://java.sun.com/products/jsp/

[44] Java Servlet Technology. Available from: http://java.sun.com/products/servlets

[45] E. K. Glenn and T. P. Stephen, "A cookbook for using the model-view controller user interface paradigm in Smalltalk-80," *J. Object Oriented Program.*, vol. 1, pp. 26-49, 1988.

[46] R. Rivest, "The MD5 Message-Digest Algorithm," 1992. Available from: http://www.ietf.org/rfc/rfc1321.txt

[47] E. Wilde, "Shared Bibliographies as Hypertext," ETH Zurich (Swiss Federal Institute of Technology) 2005.

[48] E. Wilde, "References as Knowledge Management," *Issues in Science and Technology Librarianship*, vol. 41, 2004.

[49] E. Wilde, S. Anand, and P. Zimmermann, "Management and Sharing of Bibliographies," in *Research and Advanced Technology for Digital Libraries*, 2005, pp. 479-480.

[50] Swiss Federal Institute of Technology (ETH). Available from: http://www.ethz.ch/

[51] H. Van de Sompel and O. Beit-Arie, "Open linking in the scholarly information environment using the OpenURL framework," *D-Lib Magazine*, vol. 7, 2001.

[52] E. Wilde, S. Anand, and P. Zimmermann, "ShaRef: XML-Centric Software Design," ETH Zurich (Swiss Federal Institute of Technology) 2005.

[53] Java Remote Method Invocation (RMI). Available from: http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp

[54] HSQLDB web site. Available from: http://hsqldb.sourceforge.net/

[55] I. Takashi, K. Piyanuch, H. Masahiro, and Q. Zhengyu, "ReMarkables: A Web-Based Research Collaboration Support System Using Social Bookmarking Tools," in *Proceedings of the 2006 IEEE/WIC/ACM international conference on Web Intelligence and Intelligent Agent Technology*: IEEE Computer Society, 2006.

[56] Nippon Institute of Technology web site. Available from: http://www.nit.ac.jp/english/index.html

[57] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, "Tropos: An Agent-Oriented Software Development Methodology," *Autonomous Agents and Multi-Agent Systems*, vol. 8, pp. 203-236, 2004.

[58] G. Fox and S. Pallickara, "Deploying the NaradaBrokering Substrate in Aiding Efficient Web and Grid Service Interactions," presented at Grid Computing, 2005.

[59] G. Fox, S. Pallickara, and X. Rao, "A scaleable event infrastructure for peer to peer grids," in *Proceedings of the 2002 joint ACM-ISCOPE conference on Java Grande*. Seattle, Washington, USA: ACM, 2002.

[60] S. Pallickara, H. Bulut, P. Burnap, G. Fox, A. Uyar, and D. Walker, "Support for High Performance Real-time Collaboration within the NaradaBrokering Substrate," 2005.

[61] S. Pallickara and G. Fox, "NaradaBrokering: A Distributed Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids," in *Middleware 2003*, 2003, pp. 998-999.

[62] S. Pallickara, M. Pierce, H. Gadgil, G. Fox, Y. Yan, and H. Yi, "A Framework for Secure End-to-End Delivery of Messages in Publish/Subscribe Systems," 2006.

[63] Community Grids Lab at Indiana University web site. Available from: http://communitygrids.iu.edu/index.php

[64] S. Gatziu, *Events in an active, object-oriented database system*. Hamburg: Verlag Dr. Kovac, 1995.

[65] K. R. Dittrich and S. Gatziu, "Time Issues in Active Database Systems," presented at International Workshop on an Infrastructure for Temporal Databases, Arlington, Texas, 1993.

[66] G. Liu, A. Mok, and P. Konana, "A Unified Approach for Specifying Timing Constraints and Composite Events in Active Real-Time Database Systems," 1998.

[67] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, pp. 558-565, 1978.

[68] J. F. Allen and G. Ferguson, "Actions and Events in Interval Temporal Logic," *Journal of Logic and Computation*, vol. 4, pp. 531-579, 1994.

[69] P.-s. Kam, "Discovering temporal patterns for interval-based events," 2000.

[70] C. Liebig, M. Cilia, and A. Buchmann, "Event Composition in Time-Dependent Distributed Systems," 1999.

[71] Peter R., Pietzuch R., Shand B., and B. J., "A Framework for Event Composition in Distributed Systems," presented at 4th International Conference on Middleware (MW'03), Rio de Janeiro, Brazil, 2003.

[72] R. Tolksdorf, *Laura : a coordination language for open distributed systems*. Berlin: Technische UniversitÃ¤t Berlin, Fachbereich 20, Informatik, 1992.

[73] R. Kowalski and F. Sadri, "Towards a unified agent architecture that combines rationality with reactivity," in *Logic in Databases. International Workshop LID '96 Proceedings*, D. Pedreschi and C. Zaniolo, Eds.: Springer-Verlag, 1996, pp. 137-149.

[74] P. Wyckoff, S. W. McLaughry, T. J. Lehman, and D. A. Ford, "Tspaces," *IBM Systems Journal 37*, pp. 454-474, 1998.

[75] O. Object Management Group, "The Common Object Request Broker:Architecture and Specification". Available from: http://www.omg.org/spec/CORBA/3.1/

[76] O. Object Management Group, "CORBA Services:Common Object Services Specification-Event Service Specification". Available from: ftp://ftp.omg.org/pub/docs/formal/98-07-05.pdf

[77] Sun Micro Systems, "Java AWT:Delegation Event Model". Available from: http://java.sun.com/j2se/1.3/docs/guide/awt/designspec/events.html

[78] M. Shane, B. Mark, P. Steven, and R. T. V., "An Events Syntax for XML."

[79] R. Alur and T. A. Henzinger, "Reactive Modules," *Formal Methods in System Design*, vol. 15, pp. 7-48, 1999.

[80] G. F. Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems Concepts and Design*: Addison-Wesley, 2005.

[81] B. John, B. Jean, M. Ken, and S. Mark, "Using events for the scalable federation of heterogeneous components," in *Proceedings of the 8th ACM SIGOPS European workshop on Support for composing distributed applications*. Sintra, Portugal: ACM, 1998.

[82] T. Kindberg, G. Coulouris, J. Dollimore, and J. Heikkinen, "Sharing objects over the Internet: the Mushroom approach," presented at Global Telecommunications Conference, 1996. GLOBECOM '96. 'Communications: The Key to Global Prosperity, London, UK, 1996.

[83] J. Bacon, J. Bates, R. Hayton, and K. Moody, "Using Events to Build Distributed Applications," 1995.

[84] N. Foo and P. Peppas, "Primitive Events," presented at 7th Join Australian Conference in Artificial Intelligence AI'94, Armidale,Australia, 1994.

[85] P. Pietzuch, B. Shand, and J. Bacon, "A Framework for Event Composition in Distributed Systems," in *Middleware 2003*, 2003, pp. 997-997.

[86] K. Delaney and R. Soukup, *Inside Microsoft SQL Server 2000*. Redmond, WA: Microsoft Press, 2000.

[87] T. Kyte, *Expert one-on-one Oracle*. Birmingham, Uk: WROX Press, 2001.

[88] C. Innocenti, G. Mondino, P. Regis, and G. Sandini, "Trajectory planning and real-time control of an autonomous mobilerobot equipped with vision and ultrasonic sensors," presented at Proceedings of the 1994 IEEE/RSJ/GI International Conference on Intelligent Robots and Systems IROS'94, Munich, Germany, 1994.

[89] S. Ceri and G. Pelagatti, *Distributed databases : principles and systems*. New York: McGraw-Hill, 1984.

[90] C. S. Jensen, J. Clifford, S. K. Gadia, A. Segev, and S. Richard Thomas, "A glossary of temporal database concepts," *SIGMOD Rec.*, vol. 21, pp. 35-43, 1992.

[91] P. Bonnet, J. Gehrke, and P. Seshadri, "Towards Sensor Database Systems," in *Mobile Data Management*, 2001, pp. 3-14.

[92] V. Lesser, C. L. Ortiz, and M. Tambe, *Distributed sensor networks : a multiagent perspective*. Boston: Kluwer Academic Publishers, 2003.

[93] R. Eckstein, M. Loy, and D. Wood, *Java Swing*. Sebastopol, Calif.: O'Reilly, 1998.

[94] W. O. Galitz and NetLibrary, "The essential guide to user interface design an introduction to GUI design principles and techniques," 2nd ed: New York : Wiley Computer Pub., 2002.

[95] R. Frank and NetLibrary, "Understanding smart sensors," 2nd ed: Boston : Artech House, 2000.

[96] B. Garabadu, C. Thompson, G. Lindstrom, and J. Klewicki, "Fast and Accurate NN Approach for Multi-Event Annotation of Time Series," University of Utah, 2003.

[97] A. T. Chandramohan and M. L. Henry, "Hardware and software support for efficient exception handling," in *Proceedings of the sixth international conference on Architectural support for programming languages and operating systems*. San Jose, California, United States: ACM, 1994.

[98] S. Gatziu and K. R. Dittrich, "Detecting composite events in active database systems using Petrinets," presented at Proceedings Fourth International Workshop on Research Issues in Data Engineering, Houston, TX, USA, 1994.

[99] N. Gehani, H. V. Jagadish, and O. Shumeli, "Composite Event Specification in Active Databases:Model and Implementation," presented at Proceedings of the18th International Conference on Very Large Databases, Vancouver, Canada, 1992.

[100] A. Hegyi, B. De Schutter, S. Hoogendoorn, R. Babuska, H. van Zuylen, and H. Schuurman, "A fuzzy decision support system for traffic control centers," presented at Proceedings of the Intelligent Transportation Systems, Oakland, CA, USA, 2001.

[101] M. Molina, J. Hern A, and J. E. Cuena, "A structure of problem-solving methods for real-time decision support in traffic control," *International Journal of Human-Computer Studies*, vol. 49, pp. 577-600, 1998.

[102] J. P. Bell and D. Schauder, "The WEBWORKFORCE: a learning repository to support educators, trainers and information technology courses," in *Proceedings*

of the fifth Australasian conference on Computing education - Volume 20. Adelaide, Australia: Australian Computer Society, Inc., 2003.

[103] A. Naeve, "The Knowledge Manifold:an Educational Arcitecture That Supports Inquiry-Based Customizable Forms of E-Learning," presented at The 2nd European Web-based Learning Environments Conference (WBLE 2001), Lund, Sweeden, 2001.

[104] C. Gianpaolo, N. Elisabetta Di, and F. Alfonso, "The JEDI Event-Based Infrastructure and Its Application to the Development of the OPSS WFMS," *IEEE Transactions on Software Engineering*, vol. 27, pp. 827-850, 2001.

[105] G. Fox and S. Pallickara, "The Narada Event Brokering System: Overview and Extensions," presented at Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, Nevada, USA, 2002.

[106] S. Pallickara and G. C. Fox, "A Scalable Durable Grid Event Service," in *Middleware 2003*, 2001.

[107] SERVOGrid project. Available from: http://www.servogrid.org

[108] GlobalMMCS project. Available from: http://www.globalmmcs.org

[109] Anabas Inc. web site. Available from: http://www.anabas.com/

[110] A. S. Tanenbaum and M. V. Steen, *Distributed Ssytems Principles and Paradigms*, 2002.

[111] D. Mosberger, "Memory consistency models," *SIGOPS Oper. Syst. Rev.*, vol. 27, pp. 18-26, 1993.

[112] S. V. Adve and K. Gharachorloo, "Shared Memory Consistency Models: A Tutorial," vol. 29, 1996, pp. 66-76.

[113] M. P. Herlihy and J. M. Wing, "Linearizability: a correctness condition for concurrent objects," *ACM Trans. Program. Lang. Syst.*, vol. 12, pp. 463-492, 1990.

[114] H. Attiya and J. L. Welch, "Sequential consistency versus linearizability," *ACM Trans. Comput. Syst.*, vol. 12, pp. 91-122, 1994.

[115] R. J. Lipton and J. S. Sandberg, *PRAM : a scalable shared memory*. Princeton, N.J.: Princeton University, Dept. of Computer Science, 1988.

[116] M. Dubois, C. Scheurich, and F. A. Briggs, "Synchronization, Coherence, and Event Ordering in Multiprocessors," vol. 21, 1988, pp. 9-21.

[117] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy, "Memory consistency and event ordering in scalable shared-memory multiprocessors," in *Proceedings of the 17th annual international symposium on Computer Architecture*. Seattle, Washington, United States: ACM, 1990.

[118] K. Pete, L. C. Alan, and Z. Willy, "Lazy release consistency for software distributed shared memory," in *Proceedings of the 19th annual international symposium on Computer architecture*. Queensland, Australia: ACM, 1992.

[119] N. B. Brian, J. Z. Matthew, and A. S. Wayne, "The Midway Distributed Shared Memory System," Carnegie Mellon University 1993.

[120] D. B. Terry, K. Petersen, M. J. Spreitzer, and M. M. Theimer, "The Case for Non-transparent Replication: Examples from Bayou," vol. 21: IEEE Data Engineering, 1998, pp. 12-20.

[121] D. B. Terry, A. J. Demers, K. Petersen, M. J. Spreitzer, M. M. Theimer, and B. B. Welch, "Session Guarantees for Weakly Consistent Replicated Data," in *Proceedings of the Third International Conference on Parallel and Distributed Information Systems*. Austin, TX, USA: IEEE Computer Society, 1994, pp. 140-149.

[122] K. Petersen, M. Spreitzer, D. Terry, and M. Theimer, "Bayou: replicated database services for world-wide applications," in *Proceedings of the 7th workshop on ACM SIGOPS European workshop: Systems support for worldwide applications*. Connemara, Ireland: ACM, 1996.

[123] B. Berliner, "CVS II:Parallelizing Software Dev elopment."

[124] S. Yasushi and S. Marc, "Optimistic replication," *ACM Comput. Surv.*, vol. 37, pp. 42-81, 2005.

[125] S. Weiss, P. Urso, and P. Molli, "Wooki: A P2P Wiki-Based Collaborative Writing Tool," in *Web Information Systems Engineering – WISE 2007*, 2007, pp. 503-512.

[126] G. Oster, P. Urso, P. Molli, and A. Imine, "Data consistency for P2P collaborative editing," in *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*. Banff, Alberta, Canada: ACM, 2006.

[127] P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A. M. Kermarrec, "Lightweight probabilistic broadcast," *ACM Trans. Comput. Syst.*, vol. 21, pp. 341-374, 2003.

[128] D. Alan, G. Dan, H. Carl, I. Wes, L. John, S. Scott, S. Howard, S. Dan, and T. Doug, "Epidemic algorithms for replicated database maintenance," *SIGOPS Oper. Syst. Rev.*, vol. 22, pp. 8-32, 1988.

[129] D. B. Andrew and N. Bruce Jay, "Implementing remote procedure calls," *ACM Trans. Comput. Syst.*, vol. 2, pp. 39-59, 1984.

[130] Jakarta Commons HttpClient with version 3.0.1. Available from: http://jakarta.apache.org/httpcomponents/httpclient-3.x/

[131] Document Object Model (DOM) web site. Available from: http://www.w3.org/DOM/

[132] Simple API for XML (SAX) Parser web site. Available from: http://www.saxproject.org/

[133] S. R. David and K. Balachander, "An event-based model of software configuration management," in *Proceedings of the 3rd international workshop on Software configuration management*. Trondheim, Norway: ACM Press, 1991.

[134] G. C. Fox, "Collaboration within an Event based Computing Paradigm," 2001.

[135] Dublin Core Metadata Initiative (DCMI). Available from: http://dublincore.org/

[136] SOA and Web Services. Available from: http://www.ibm.com/developerworks/webservices

[137] Web Services Description Language (WSDL). Available from: http://www.w3.org/TR/wsdl

[138] Simple Object Access Protocol (SOAP) 1.1. Available from: http://www.w3.org/TR/soap/

[139] Hypertext Transfer Protocol (HTTP) 1.1. Available from: ftp://ftp.isi.edu/in-notes/rfc2616.txt

[140] FILE TRANSFER PROTOCOL (FTP). Available from: http://tools.ietf.org/html/rfc959

[141] Google Scholar. Available from: http://scholar.google.com

[142] Windows Live Academic. Available from: http://academic.live.com

[143] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *The VLDB Journal The International Journal on Very Large Data Bases*, vol. 10, pp. 334-350, 2001.

[144] H. Kreger, "Web Services Conceptual Architecture (WSCA 1.0)," 2001.

[145] F. E. Redmond, *DCOM : Microsoft Distributed Component Object Model*. Foster City, CA: IDG Books Worldwide, 1997.

[146] Open Source Version Control web site. Available from: http://www.nongnu.org/cvs/

[147] Subversion web site. Available from: http://subversion.tigris.org/

[148] H. T. Kung and T. R. John, "On optimistic methods for concurrency control," in *Proceedings of the fifth international conference on Very Large Data Bases - Volume 5*. Rio de Janeiro, Brazil: VLDB Endowment, 1979.

[149] S. Chengzheng and C. David, "Consistency maintenance in real-time collaborative graphics editing systems," *ACM Trans. Comput.-Hum. Interact.*, vol. 9, pp. 1-41, 2002.

[150] L. Jiang, L. Xiaotao, S. Prashant, and R. Krithi, "Consistency Maintenance In Peer-to-Peer File Sharing Networks," in *Proceedings of the The Third IEEE Workshop on Internet Applications*: IEEE Computer Society, 2003.

[151] R. Jonathan, F. Sarah, and V. Sankar, "Consistency management for distributed collaboration," *ACM Comput. Surv.*, vol. 31, pp. 13, 1999.

[152] V. Jurgen, V. JiRgen, G. Werner, C. Li-Te, and M. Michael, "Consistency Control for Synchronous and Asynchronous Collaboration Based on Shared Objects and Activities," *Comput. Supported Coop. Work*, vol. 13, pp. 573-602, 2004.

[153] G. Werner, V. Jurgen, C. Li-Te, and M. Michael, "Supporting activity-centric collaboration through peer-to-peer shared objects," in *Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work*. Sanibel Island, Florida, USA: ACM Press, 2003.

[154] L. Rui, L. Du, and S. Chengzheng, "A Time Interval Based Consistency Control Algorithm for Interactive Groupware Applications," 2004.

[155] XML Path Language (XPATH). Available from: http://www.w3.org/TR/xpath

[156] JTIDY with version 04aug2000r7. Available from: http://jtidy.sourceforge.net/

[157] JDOM with version 1.1. Available from: http://www.jdom.org/

[158] Java 2 SDK, Standart Edition with version 1.5. Available from: http://java.sun.com/

[159] Internet Documentation and Integration of Metadata (IDIOM) Project web site. Available from: http://gf16.ucs.indiana.edu:54571/IDIOM/login.jsp

[160] D. E. Atkins, K. K. Droegemeier, S. I. Feldman, H. Garcia-Molina, M. L. Klein, D. G. Messerschmitt, P. Messina, J. P. Ostriker, and M. H. Wright, "Revolutionizing Science and Engineering Through Cyberinfrastructure. Report of the National Science Foundation Blue-Ribbon Advisory Panel on Cyberinfrastructure," 2003.

[161] T. O'Reilly, *What is Web 2.0: Design patterns and business models for the next generation of software*, 2005. Available from: http://www.oreillynet.com/lpt/a/6228

[162] C. Anderson, *The Long Tail: Why the Future of Business Is Selling Less of More*: Hyperion, 2006.

[163] IBM WebSphere Session Management. Available from: http://www.informit.com/articles/article.asp?p=332851&rl=1

[164] A. M. Scott, K. Richard, L. Matt, and S. Craig, "PubsOnline: open source bibliography database," in *Proceedings of the 33rd annual ACM SIGUCCS conference on User services*. Monterey, CA, USA: ACM Press, 2005.

[165] B. Hasan, P. Shrideep, and F. Geoffrey, "Implementing a NTP-based time service within a distributed middleware system," in *Proceedings of the 3rd international symposium on Principles and practice of programming in Java*. Las Vegas, Nevada: Trinity College Dublin, 2004.