Information Systems 55 (2016) 20–36

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

A novel digital information service for federating distributed digital entities

Ahmet F. Mustacoglu [a,*], Geoffrey C. Fox [b,c]

[a] TUBITAK BILGEM – The National Research Institute of Electronics and Cryptology (UEKAE), Turkey
[b] School of Informatics and Computing, Indiana University, Bloomington, IN, USA
[c] Community Grids Lab, Indiana University, Bloomington, IN, USA

ARTICLE INFO

Article history:
Received 30 June 2015
Received in revised form
27 July 2015
Accepted 28 July 2015
Available online 8 August 2015

Keywords:
Web services and Service-Oriented computing
SOA
Web 2.0
Data management
Information retrieval and management
Federation and unifications

ABSTRACT

We investigate the performance and the scalability metrics of a Digital Information Service framework that is used for unifying and federating online digital entities by retrieving and managing information located on the web. The Digital Information Service consists of tools and web services for supporting Cyberinfrastructure based scientific research. This system supports a number of existing online Web 2.0 research tools (social bookmarking, academic search, scientific databases, journal and conference content management systems) and aims to develop added-value community building tools that leverage the management and federation of digital entities and their metadata obtained from multiple services. We introduce a prototype implementation and present its evaluation. As the results indicate, the proposed system achieves federation and unification of digital entities coming from different sources with negligible processing overheads.

1. Introduction

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 with respect to 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 kinds of resources. Flickr, for example, allows the tagging and sharing of photos, del.icio.us the tagging and sharing of bookmarks, BibSonomy, CiteULike and Connotea the tagging and sharing of scholarly publications, YouTube the tagging and sharing of video, and 43Things the tagging and sharing of goals in private life. Some of these tools may not survive in the future, for example, Connotea ended operation in March 12, 2013 due to the growing problems with spam and associated service outages. Note that this type of cases can be thought of as an advantage since our proposed system stores data coming from external services locally and the stored data can be retrieved and exported into supported file formats (.txt etc.). Furthermore, in the case of these tools are not operating continuously (e.g., system down status of few days) our system would continue to operate perfectly. The proposed system stores the data in its own database hence the data

* Corresponding author.
E-mail addresses: afatih.mustacoglu@tubitak.gov.tr (A.F. Mustacoglu), gcf@indiana.edu (G.C. Fox).

http://dx.doi.org/10.1016/j.is.2015.07.007
0306-4379/

Table 1
Status of Web-based popular academic/non-academic services.

Tool name	Academic (A)/non-academic (NA)	Start–end date	Users/data as of now (valid as of 02.01.2015)	Current status
Citeulike	A	11.2004–current	Around 7.4 million articles	Active
Connotea	A	12.2004–03.2013	No information	Retired
BibSonomy	A	Early 2006	Around 3.4 million links	Active
Google Scholar	A	11.2004–current	All scholarly literature	Active
Microsoft Academic Search	A	11.2009–current	Over 4.1 million papers	Not updated
ResearchGate	A	05.2008–current	Over 3 million members	Active
IEEE Xplore	A	02.2000–current	Over 3 million documents	Active
CiteSeerX	A	1998–current	Over 2 million documents	Active
Science Direct	A	1997–current	Over 11 million articles	Active
DBLP	A	09.2006–current	Over 2.3 million articles	Active
ACM DL	A	1997–current	391,000 full text articles	Active
Mendeley	A	07.2013–current	Over million documents	Active
Delicious	NA	09.2003–current	Around 1 billion links	Active
YouTube	NA	02.2005–current	Over 1 billion visitor users/Over 6 billion hours of video watching per month	Active
Flickr	NA	02.2004–current	87 million users/over 6 billion images	Active
43Things	NA	01.2005–current	Over 3 million users	Active
Pinterest	NA	03.2010–current	70 million users worldwide	Active
Facebook	NA	02.2004–current	1.2 billion	Active

can be retrieved at any time as needed: the data is updated in a system adaptive way, and the data in these services are mirrored in our proposed system with a frequency that can be determined adaptively. Table 1 explains the status of popular web-based academic and non-academic tools as of now.

The some of the services that are marked as *academic* and *nonacademic* in Table 1 are directly integrated into the our proposed system (citeulike, Connotea, GoogleScholar, GoogleScholar Advance, Microsoft Academic Search, Delicious) and the others are a natural candidate for implementing our algorithm. Note that Microsoft Academic Search has *active* status but its database is not updated since 2012.

As the web-based academic or nonacademic tools enabling storing, tagging and sharing documents have gained popularity, an emerging need has appeared for supporting these tools by using their existing services via Web Service wrappers with added capabilities. To address this challenges, an ideal architecture should meet the following requirements: a) *uniformity*: the architecture should support one-to-many services among information resources and their communication protocols; b) *federation*: the architecture should present a federation capability where different services belonging to different annotation resources on the web can interoperate with each other; c) *interoperability*: the architecture should be interoperable with different kinds of clients on the web; d) *performance*: the architecture should search/retrieve/store metadata for scholarly publications with negligible processing overheads; e) *persistency*: the architecture should be able to back-up metadata about digital records without affecting the system performance; and f) *fault tolerance*: the architecture should be distributing metadata describing a digital content and managing redundancy of metadata about digital entities in acceptable rates. Fig. 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 retrieval and management in a consistent way.

We propose a Digital Information Service framework that improves the previous work [23] in great detail for reconciling distributed digital entities that addresses the challenges of discovering, retrieving, sharing and managing distributed data located on web-based systems in a Service Oriented Architecture where communications are provided through the Web Service technology. The proposed system provides users with ability to access their data even if a web-based system retires and discontinues its service in the future.

In this study, we present the semantics and the architectural design of the centralized Digital Information Service. We introduce a prototype implementation called IDIOM (Internet Documentation and Integration of Metadata) of

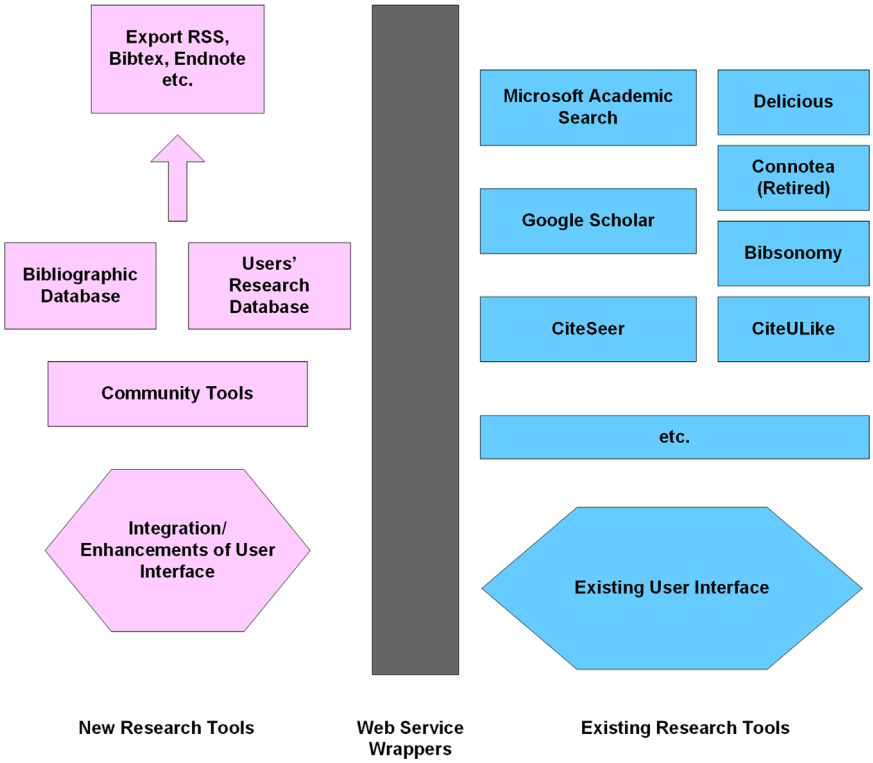

Fig. 1. Research tools with added capabilities for Sharing and Managing Scientific Documents.

this architecture and present its performance evaluation. As the main focus of this research is information federation in online digital information systems, we discuss unification, federation, interoperability, and performance aspects and leave out distribution and fault-tolerance aspects of the proposed system. The main novelty of this study is that it describes an architecture, implementation, and evaluation of the Digital Information Service that supports both distributed and centralized paradigms and handles both dynamic, small-scale and static, large-scale metadata by utilizing event-based infrastructure and consistency maintenance mechanism [21]. This novel approach unifies different implementations of research tools for scholarly publications to provide a common access interface to different kinds of metadata [4]. It also provides federation of information among the scholarly publications tools for digital entities, so that they can share or exchange metadata with each other [30]. This study should inspire the design of other information systems along with similar metadata management requirements.

The organization of the rest of this paper is as follows. Background information relevant to this study is given in Section 2. Section 3 provides an overview of the proposed Digital Information Service. Section 4 presents the semantics of the Digital Information Service. Section 5 describes the architectural design and the prototype implementation of the system in details. Section 6 evaluates the performance and the scalability test results for the prototype implementation of the Digital Information Service framework. Last, we conclude with some final remarks and future work in Section 7.

2. Background

We overview the event systems and the consistency maintenance issues for distributed systems that are crucial for the proposed Digital Information Service framework in the following sub-sections.

2.1. 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. For example, NaradaBrokering [10,11,25] system implements publish–subscribe paradigm and it is an open-source event-based messaging infrastructure developed by the Community Grids Lab at Indiana University.

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 [8,12,20]. 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 by [17], and unique timestamp values are assigned automatically to each event by the system.

The second approach defines an event as occurrence as an interval in time [2,13,19,26]. 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 the ending points of an interval.

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

Discussion: in our research, we have chosen to use the first approach to define events due to its suitability to our design of the proposed Digital Information Service 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 our prior work. The assigned time stamp values provide us with ability to sort events based on their occurrences. Our proposed system can generate any version of a final document by using the sorted events. Furthermore, the proposed system uses time stamp values for consistency maintenance described in detail in Section 4.

2.2. Event representation

According to [15,32,33], events are described 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 attributes. Events are constructed in the form of tuple structure and delivered to external entities that are listening to 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 vary based on the domain of each system. Messages in event system portray a tuple structure and generic tuples composed of:

- Unique Event Id.
- Event attributes that carry 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 carry 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, Java AWT delegation Event Model, DOM interfaces for tuple representation. In database programming, events are stored as tuples in the form of record structures composing the event histories.

Every system has a response unit to the state changes coming from various environments to handle the changes. Reactive applications depend on the data that describe the current state of their environment due to changes. Each application continuously checks any state changes happening in their environment to obtain 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, resulting in reducing the computational load. 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 [3,15].

In distributed event-based systems, multiple objects at different locations can be notified by events that could take place at any of these 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 [7]:

- Heterogeneous: when event-based systems are used for communication between distributed objects, different components that are 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 [5].
- Asynchronous: event generating objects send notifications to all objects that subscribe to them so that publisher does not need to synchronize with the subscriber objects. Project Mushroom described in detail by [14] is a distributed event-based system that supports collaborative work.

Discussion: in this study, each event has a unique event id, and we have distinguished our events as major and minor events. We have defined our events as a time-stamped action on a digital document. In our study, we have unified and federated heterogeneous annotation tools to communicate with each other via event-based infrastructure and Web Service technology. Moreover, we have integrated search and web-based academic search tools that are used for retrieving and collecting data and metadata from internet into the proposed system. We did not

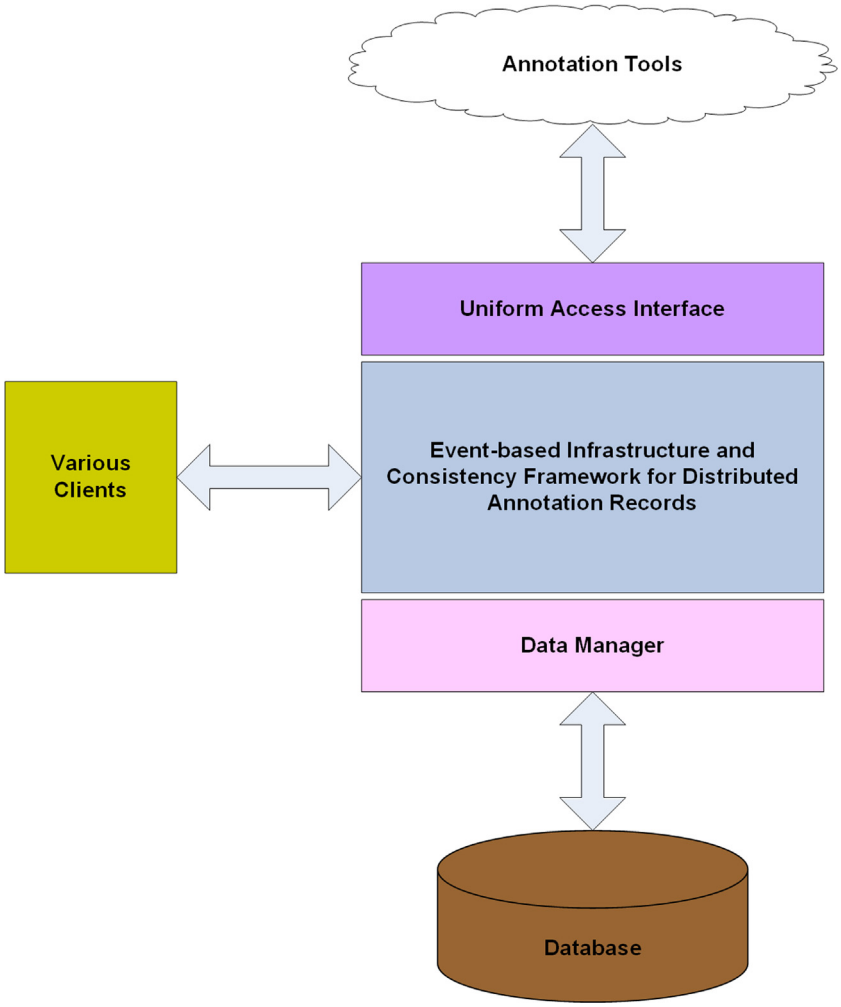

Fig. 2. General architectural design of a Digital Information Service.

use publish–subscribe paradigm to disseminate updates since the integrated annotation tools do not support publish–subscribe mechanism. However, any application that requires and supports publish–subscribe mechanism, then a broker address and a 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 the topic. Finally, our update propagation falls into unicast communication technology that requires sending updates to each annotation tool separately by the system not the underlying mechanism.

2.3. Consistency maintenance

Consistency is an important issue in distributed systems. Consistency means that all copies of a same document should be the same. When one copy is updated, then it must be ensured that all copies are updated as well. According to [31], consistency models can be classified into two groups: 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 sub-sections.

2.3.1. Data-centric consistency models

A consistency model is an agreement 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 provides better results than the one with minor restrictions do [31]. More information on consistency models is explained in detail by [1,22]. Tanenbaum classifies

data-centric consistency models into seven sub-categories: a) Strict Consistency; b) Linearizability and Sequential Consistency; c) Casual Consistency; d) FIFO Consistency; e) Weak Consistency; f) Release Consistency; and g) Entry Consistency [31].

2.3.2. Client-centric consistency models

In the previous section, we have overviewed and summarized data-centric consistency models that are all about providing a system wide 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: a) Eventual Consistency; b) Monotonic Reads; c) Monotonic Writes; d) Read Your Writes; and e) Writes Follow Reads [31].

Discussion: the consistency framework of the proposed Digital Information Service falls into a client-centric consistency model, and the implementation protocol is the replicated-write protocol because updates can be originated from several replicas. In this research, the optimistic replication approach [16,29] has been adopted to ensure eventual consistency between replicas. The consistency in our system is made possible by updating the relevant databases with a period of 24 h. Shorter period would entail more up to date database entries but degrade the system/network performance adversely. Details can be found in Section 4.

3. Digital Information Service

We designed and built a novel Information Service called Digital Information Service to provide an ideal approach to unify and federate major web-based annotation/search tools, support collaboration, retrieve, represent and manage content of scientific documents coming from various sources in a flexible fashion. Digital Information Service forms an add-on architecture that interacts with the various social networking tools and unifies them in a higher-level system. In other words, it provides a unifying architecture, where one can assemble metadata instances of different web-based information services. We built a prototype implementation called Internet Documentation and Integration of Metadata (IDIOM) that showed that the proposed Digital Information Service achieves unification and federation of the three academic publication management tool implementations, namely, Connotea, Delicious and Citeulike, and support their communication protocols. Furthermore, the prototype implementation also supports ability to use major academic search tools (Microsoft Academic Search and Google Scholar etc.) to collect metadata and store them into a local system. We also showed that the Digital Information Service achieves information federation by utilizing a global schema called Merged Schema. The merged schema consists of annotation tools' schemas, academic search tools' schemas, Dublin Core Metadata Initiative schemas and BibTex schemas. With these capabilities, the proposed Digital Information Service enables implementations of different digital metadata management and academic search tools to interact with each other and to share each other's metadata. We discuss the semantics and architecture of the proposed Digital Information Service in the following sections.

Table 2
Stored metadata comparison in major annotation tools.

Stored metadata	Citeulike	Delicious	Connotea (discontinued)
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	✓		
Organization	✓		
Publisher	✓		
Address	✓		
School	✓		
Series	✓		
Bibtex key	✓		
Abstract	✓		
Display title			✓
Tags	[a]	✓	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.
[a] Adds "no-tag" footnote.

4. Semantics of the Digital Information Service

In this section, we discuss three core underlying mechanism of the proposed Digital Information Service: *uniform access interface*, *event-based infrastructure* and *consistency maintenance*. General architectural design of the Digital Information Service appears in Fig. 2.

4.1. Unified Access Interface

The Digital Information Service system supports one to many annotation/academic search tools interactions and

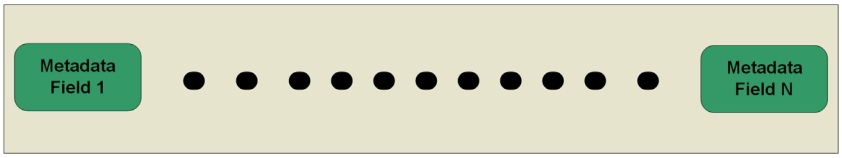

Fig. 3. Content of a digital entity.

their communication protocols by utilizing a Unified Access Interface. The Uniform Access Interface presents a common access interface to the integrated web-based annotation and academic search tools. Namely, the Uniform Access Interface imports APIs of the supported annotation and academic search tools so that they are all accessible from a common interface. This way, the proposed system unifies different annotation/academic search tools under one hybrid system.

To meet the federation requirements, the Digital Information Service framework presents a federation capability where different annotation/academic search tools and their services can be federated in metadata instances. To enable this capability, we introduce a global schema for annotation/search tools by integrating different annotation/search tools data models. The global schema for annotation/search tools provides a common platform to enable interaction between the annotation/search tools, and it represents a merged schema from the federated annotation/search tools by integrating their schemas into one. Schema integration is a functionality of providing a unified representation of multiple data models [27]. To meet the comprehensive metadata field requirements, the Digital Information Service infrastructure supports various metadata fields to represent the complete metadata about a scholarly publication. Supported metadata fields by the proposed study are compatible with the one that specified by the Dublin Core Metadata Initiative and BibTex. Table 2 portrays the stored metadata comparison in Connotea, Citeulike, and Delicious annotation tools that are integrated with the IDIOM.

4.2. Event-based infrastructure and consistency maintenance

The event-based infrastructure utilizes the use of event concept as its building block 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. According to this concept, the content of scientific documents originating from various sources is represented as events. Events constitute the base atomic 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 [28]. Storing all the events about an object enables the actions on this object to be reviewed and undone [9]. An event may also be defined as an action with a time stamp and a message [25]. In our event-based infrastructure of the proposed Digital Information Service, 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:

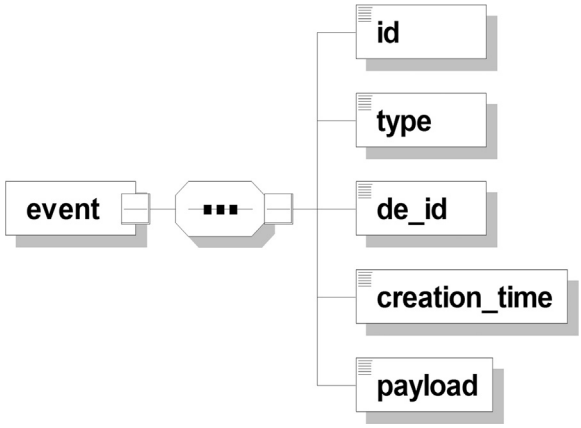

Fig. 4. Minor event parameters.

insertion of a new digital entity (DE), which is a collection of metadata representing a scholarly publication represented in Fig. 3 into the system or deletion of an existing digital entity from the system is considered a major event; updates/modifications to existing digital entities are considered minor events. 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 an XML message to the necessary modules. Schema of parameters of a minor event is depicted in Fig. 4. 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: 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) and 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 collection of events is depicted in Fig. 5. As it is seen in 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. Details about how a document is formed from major and minor events are explained in the sub-section of Event Processing Engine.

The supported annotation tools by a Digital Information Service hold metadata about scholarly publications forming a digital record being referred as a distributed annotation

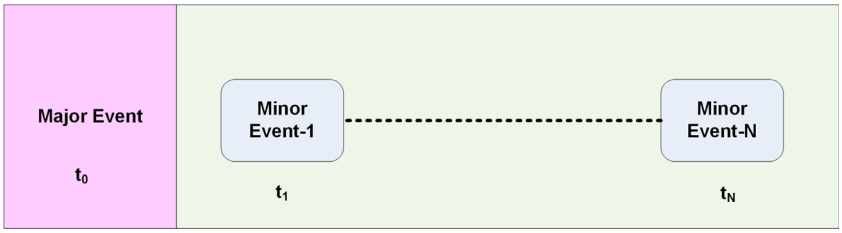

Fig. 5. Document representation as an event.

records (DARs). Furthermore, each integrated annotation tool acts as a replica to the Digital Information Service. Consistency maintenance mechanism of the Digital Information Service has been designed to ensure eventual consistency between distributed annotation records that are stored at the integrated annotation tools and a primary copy of each DAR that is located in a local database of the proposed Digital Information Service. 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 adopted the optimistic replication approach [16,29] to ensure eventual consistency between replicas. In our proposed study, update propagations are carried out through pull and push based approaches. The push approach enforces consistency model on primary copies of DARs located in a central 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 existing DARs on their site. However, the pull approach is a time-based consistency control approach [18]. Each supported annotation tool and DARs located on the annotation tools is periodically checked for any updates. Collecting updates from supported annotation tools require: 1) finding the primary copy of each replica record by using our duplicate detection algorithm; 2) comparing each replica record with its primary copy to figure out modifications if there is any. After identifying the updates, the next step is to apply them to their primary copies and disseminate them to all replicas located on the annotation tools. If there is any concurrent update on a shared document, then the concurrent updates are handled based on optimistic approach as defined by [31]. The integrated annotation tools do not support publish–subscribe paradigm forcing the Digital Information Service to use unicast communication to propagate updates to replicas. However, any application that require and support publish–subscribe concept, then broker address and topic can be defined in a property file to provide updates via publish–subscribe methodology by connecting to the broker and subscribing a topic. The Digital Information Service also supports roll-back ability to help maintain consistency due to the nature of the proposed event-based mechanism. It basically allows users to roll-back to a previous state at any time. It is also 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. The event-based infrastructure executes its duplicate detection algorithm to decide whether two given digital entity is similar or not with a defined threshold value. The duplicate detection algorithm works based on hashing the available metadata fields of a given document including URL, title, authors and publication venue. As the main focus of this paper is to discuss information federation in academic search/annotation tools, a detailed discussion on concurrent access to shared document and duplicate detection aspects of the system is omitted here.

Finally, the Digital Information Service has a well-defined update model that is built on the event-based structure to provide flexible choices to users. The update model uses events for applying 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.

So, the update model supports the above choices 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.

4.3. 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 the database of the Digital Information Service. Having done that, the Event Processing Engine processes 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. In other words, by using the initial metadata, which is a major event, of a digital entity and by applying the dataset(s) on top of it, one can retrieve any version of a DE. Hence, in case of an error or users' request, the proposed architecture supports to restore the system to a previous safe state by using the related dataset for that state.

Fig. 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 the event-based infrastructure of the Digital Information Service, 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

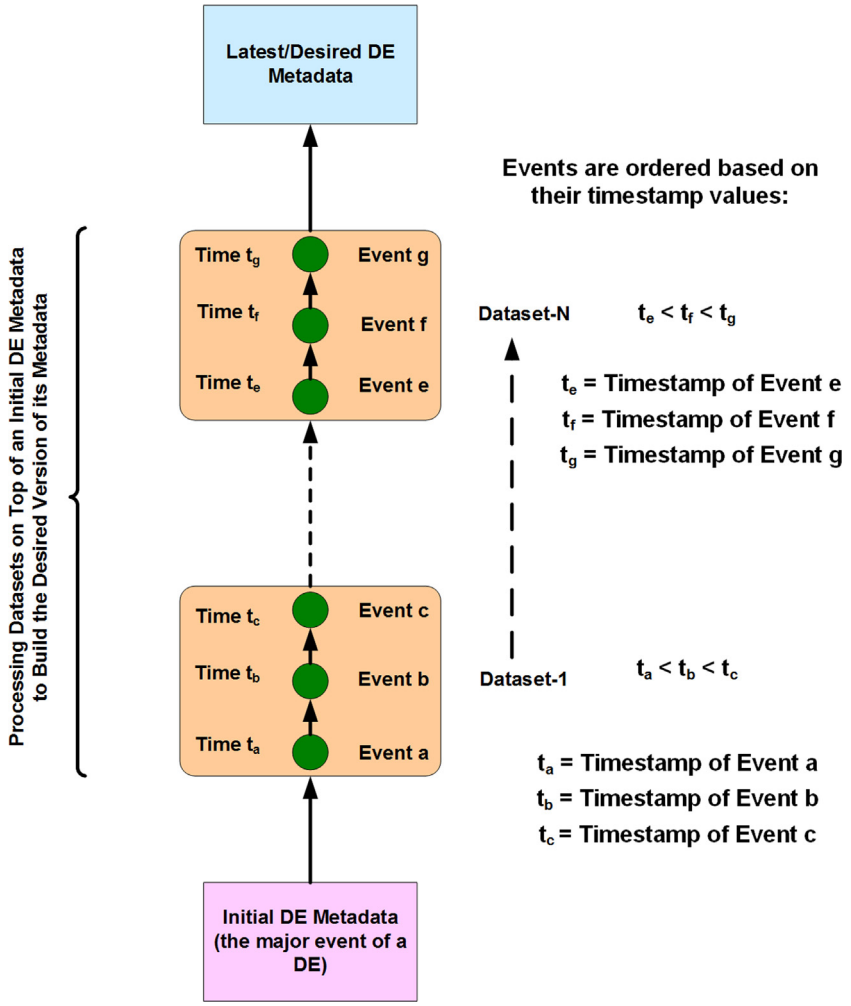

Fig. 6. Forming a document from its events.

entity metadata. During the application process, we apply each dataset and its associated events in the increasing order of their creation time.

As depicted in Fig. 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:

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

5. Architecture

The Digital Information Service is an add-on system that interacts with major academic search/annotation tools and unifies them in a higher-level architecture. Fig. 7 illustrates the detailed architectural design of the prototype implementation of the Digital Information Service. The annotation/academic search tools interact with the system through the uniform access interface. The prototype implementation IDIOM supports XML API for Connotea, Citeulike, Delicious annotation tools, Google Scholar and Microsoft Academic search engines, and the Merged Schema (combines different schemas for representing the metadata of scholarly publications into one global schema for federation of web-based annotation tools). This layer is designed as generic as possible so that it can support one-to-many XML APIs, as the new web-based tools are integrated with the system.

The IDIOM prototype implementation consists of five main layers: a) the client layer; b) the service 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, 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's Web Services, and the client layer communicate with the Server layer over the HTTP protocol through SOAP messages encapsulating WSDL-formatted objects. The Server layer consists of several 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

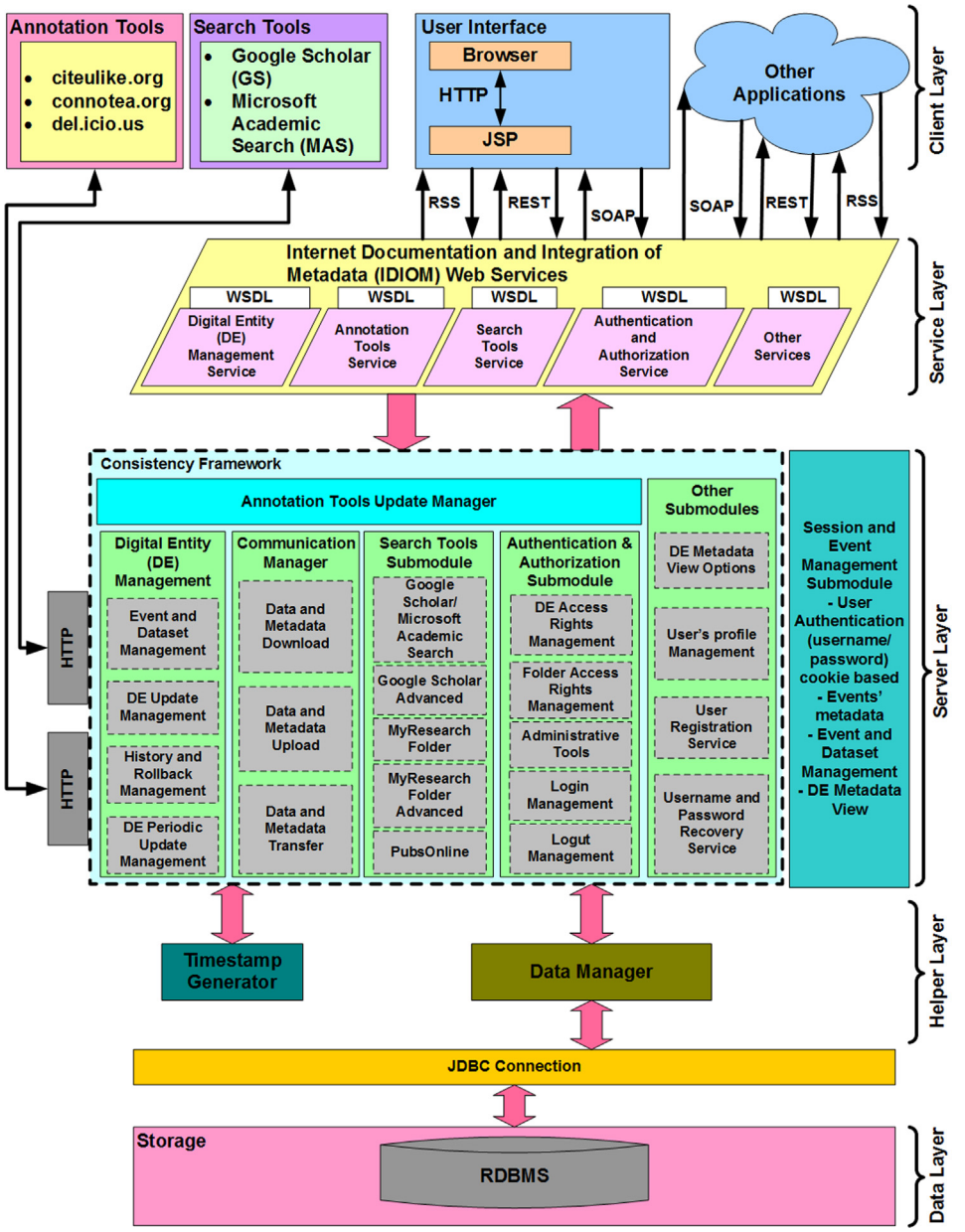

Fig. 7. Internet Documentation and Integration of Metadata (IDIOM) Architecture.

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.

Annotation tools are the integrated annotation tools into the IDIOM system to store replica copies of the primary copies referred as DEs stored in a MySQL system database of the IDIOM system. 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, the IDIOM system unifies and federates Connotea, CiteULike, and Delicious annotation tools.

The 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.

The goal of session and event management sub-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. From the moment user logged in the IDIOM system, user credentials, any changes made to a DE, and view options for metadata

Table 3
Summary of cluster nodes.

	Cluster nodes	
	cluster1.ucs.indiana.edu	**cluster2.ucs.indiana.edu**
Processor	Intel® Xeon™ CPU (E5345 2.33 GHz)	Intel® Xeon™ CPU (E5345 2.33 GHz)
RAM	8 GB (each node)	8 GB total
OS	GNU/Linux (kernel release 2.6.9-5.ELsmp)	GNU/Linux (kernel release 2.6.9-5.ELsmp

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 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; 2) implementing the Events and Dataset Management services, and providing a service to view detailed information about a DE by utilizing Event Processing Engine; 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; 4) providing an access to the history of a DE and rollback mechanism, from its entry into the IDIOM system to present; 5) providing a service to retrieve and apply updates belonging to other users on their DEs by the Periodic Update Management service.

The Communication Manager 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; 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 native libraries.

The search tools submodule provides services and interfaces to the web-based search tools including Google Scholar, Google Scholar Advanced, and Microsoft Academic Search. It also provides services for local folder search and integrates the PubsOnline software [24] – "an open source tool for management and presentation of databases of citations via the Web" – into the IDIOM system and providing an interface for searching the logical folders of the IDIOM system database.

The User Registration, The Username and Password Recovery, The User's Profile Management, and The 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.

The Timestamp Generator module is responsible for producing unique 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 the Event Processing Engine by their ordered timestamps. 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 [6]. This service provides synchronized timestamp values by synchronizing the distributed machine clocks with atomic time servers available across the universe.

The Data Manager is responsible for executing the coming requests on data items. The Data Manager uses JDBC connection to connect to MySQL system database.

6. The evaluation of the proposed system

We performed extensive series of measurements to evaluate the prototype implementation of the proposed architecture and investigate its practical usefulness in real life applications.

6.1. Testing environment

We tested the IDIOM prototype implementation by using gf12–15 and gf16 Linux machines that are part of clusters (cluster1 and cluster2) located at Community Grids Laboratory at Indiana University. We have run our client programs on gf12–gf15 Linux machines, we have deployed the IDIOM system on gf16 Linux machine, and we have installed our database on gf16 Linux machine. Summary of these machine configurations are given in Table 3.

In our general experiments methodology, we have used single-threaded and multi-threaded client programs. The IDIOM system 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 the IDIOM system 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 as a container with version 5.0.28 and Apache Axis technology for Web Service technology with version 1.2. 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

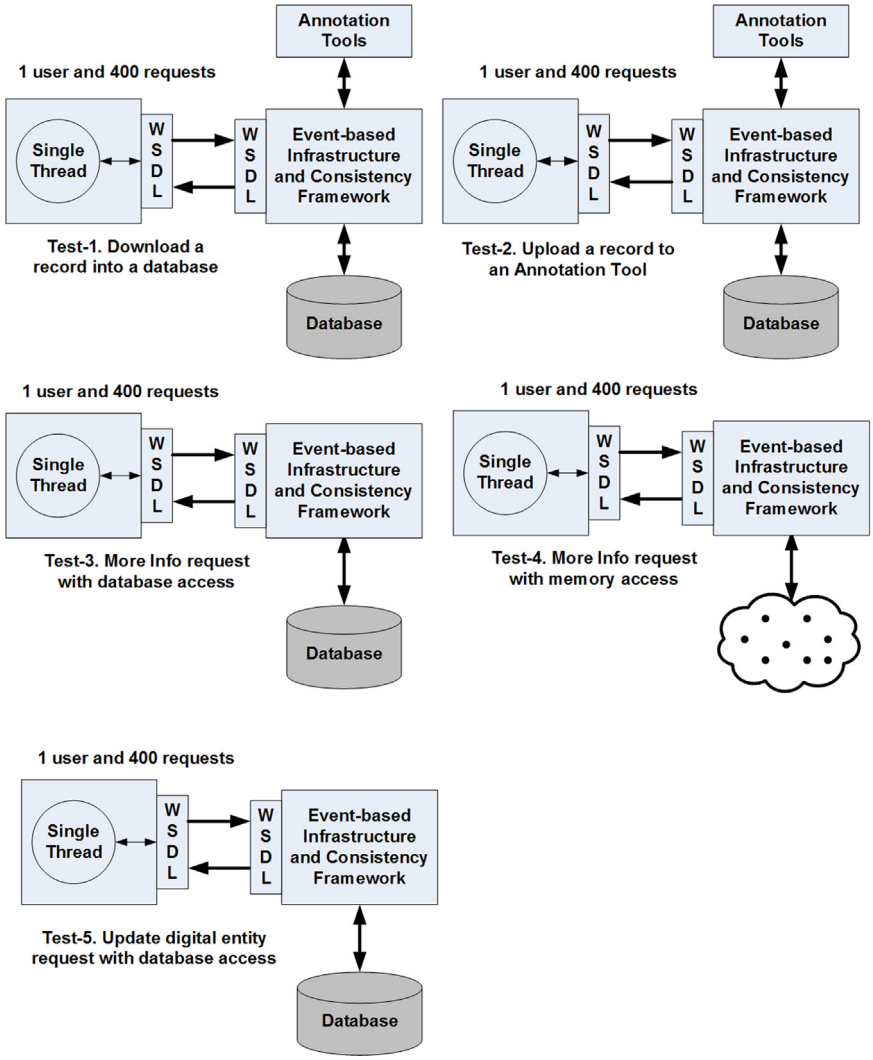

Fig. 8. Testing cases for system responsiveness experiment.

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 the IDIOM 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 IDIOM 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. Fig. 8 shows the design of these experiments.

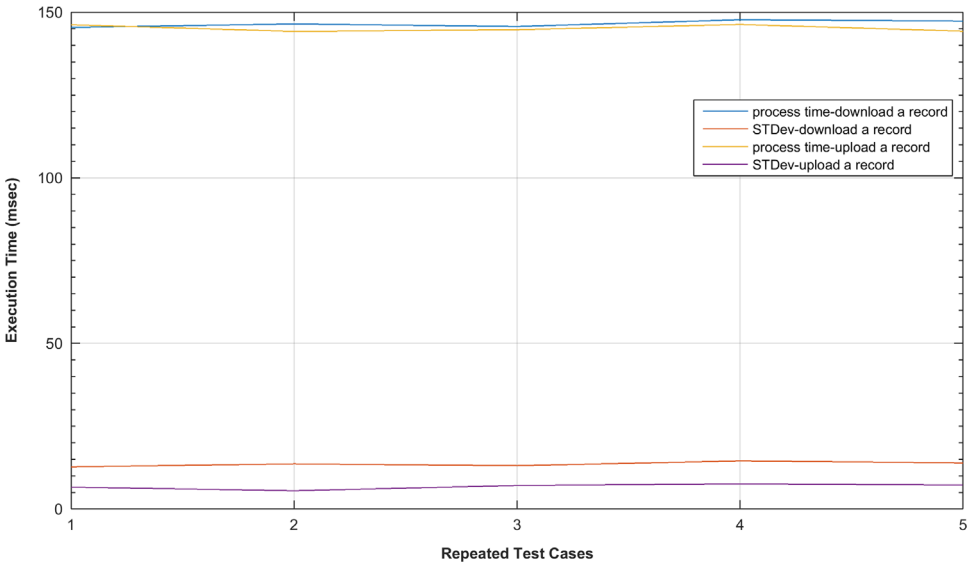

Fig. 9. Depiction of downloading and uploading a record.

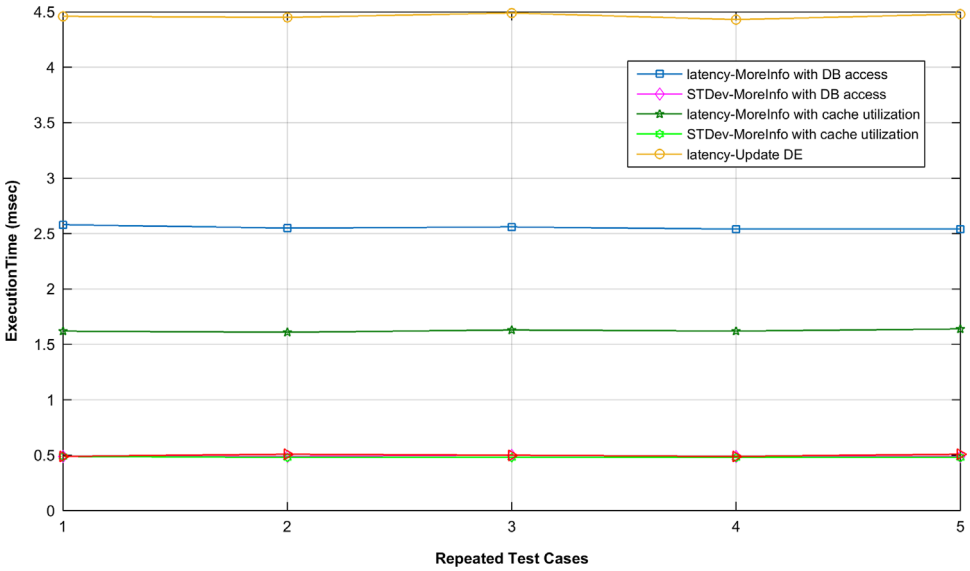

Fig. 10. Latency and STDev values for update DE and More Info standard operation (with DB and memory utilization).

6.2.1. System responsiveness experiment results

We conduct experiments where we investigate the base performance of the proposed system. Figs. 9 and 10, and Table 4 and Table 5, represent basic responsiveness results of our system. 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 prototype IDIOM system. Furthermore, the IDIOM 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 database and annotation tools (replicas).

6.3. Scalability experiment

The primary interest in doing this experiment was to investigate the scalability of the IDIOM prototype implementation. We conducted three testing cases and tried to answer the following research questions: a) how well does the system

Table 4
Statistics of the experiment depicted in Fig. 9.

Repeated test cases	1	2	3	4	5
Download Process time (m s)	145.44	146.49	145.72	147.77	147.37
Download STDev	12.74	13.64	13.09	14.54	13.94
Upload Process time (m s)	146.24	144.23	144.75	146.33	144.3
Upload STDev	6.61	5.52	7.11	7.6	7.24

Table 5
Statistics of the experiment depicted in Fig. 10.

Repeated test cases	1	2	3	4	5
Latency-MoreInfo with DB access	2.58	2.55	2.56	2.54	2.54
STDev-MoreInfo with DB access	0.49	0.49	0.50	0.49	0.49
Latency-MoreInfo with cache utilization	1.62	1.61	1.63	1.62	1.64
STDev-MoreInfo with cache utilization	0.49	0.48	0.48	0.48	0.48
Latency-Update DE	4.46	4.45	4.49	4.43	4.48
STDev-Update DE	0.49	0.51	0.50	0.49	0.51

perform 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?; c) how well does the system perform 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 IDIOM system. We have completed this test case by increasing the message rate/s 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 IDIOM while message rate per second are increased. The designs of these testing cases are depicted in Fig. 11.

6.3.1. Scalability experiment results

Based on the results depicted in Fig. 12, we determined that concurrent inquiry requests may be well responded by the IDIOM prototype implementation without any error. According to the experiment result, we identified that IDIOM's 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 More Info with DB access, after around 2068 inquiry messages per second for More Info 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 the following sub-section. Experiment results are depicted in Fig. 12.

6.4. 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.

6.4.1. 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: 3738 bits/message * 3693 message/s = 13.8 Mbits/s.
- Message size in More Info request is 879 bytes. Message size in bits 879 bytes * 8 bits = 7032 bits. A total network is needed at the threshold value is: 7032 bits/message * 2068 message/s = 14.5 Mbits/s.
- Message size in Update metadata request is 3700 bytes. Message size in bits 3700 bytes * 8 bits = 29,600 bits. A total network is needed at the threshold value is: 29,600 bits/message * 533 message/s = 15.8 Mbits/s.

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

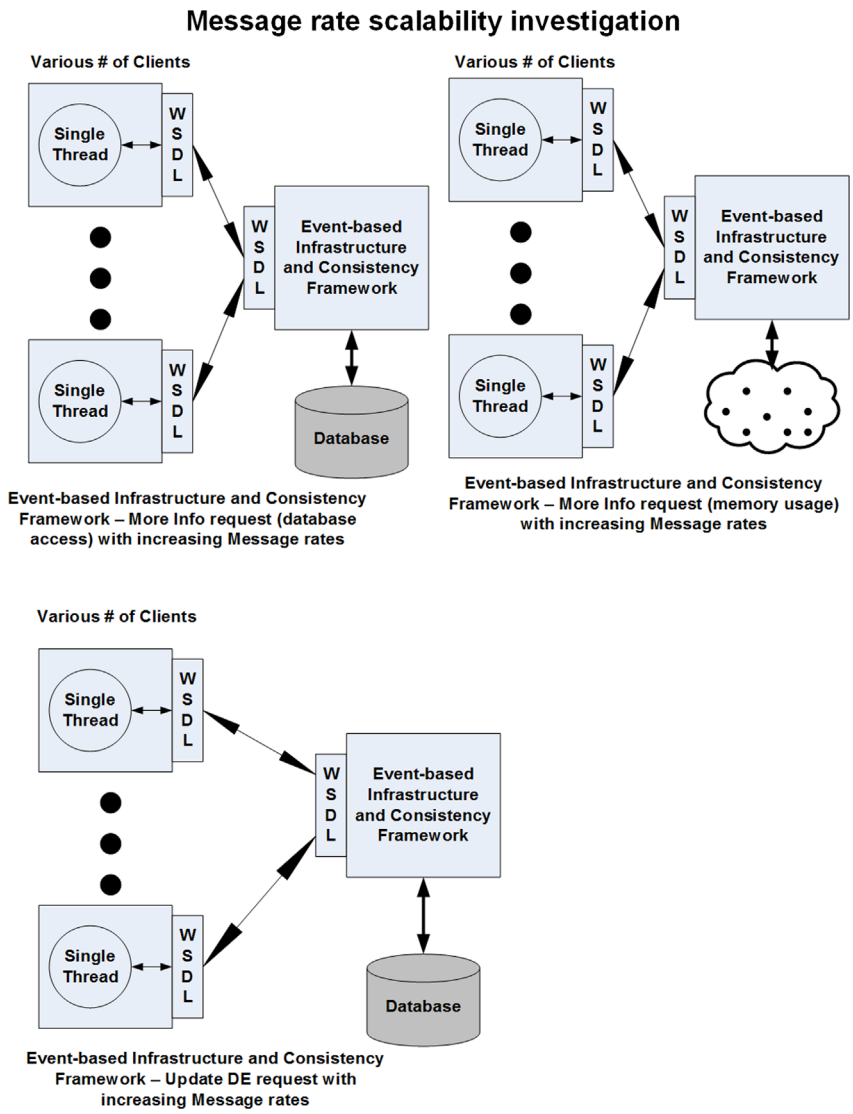

Fig. 11. Testing cases of scalability experiment for More Info and update DE requests.

6.4.2. 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.

6.4.3. 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. Fig. 13 represents our investigation results.

Finally, we have concluded based on the results that we obtained in Fig. 13 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 empty service call measurements.

7. Conclusion and future work

In this research, we introduced a novel Digital Information Service architecture for a Collaborative Framework for Distributed Digital Entities that supports handling metadata coming from different sources. The proposed Digital Information Service provides unification, federation, and interoperability of different annotation and academic search tools by using Web Services technology. The proposed study 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

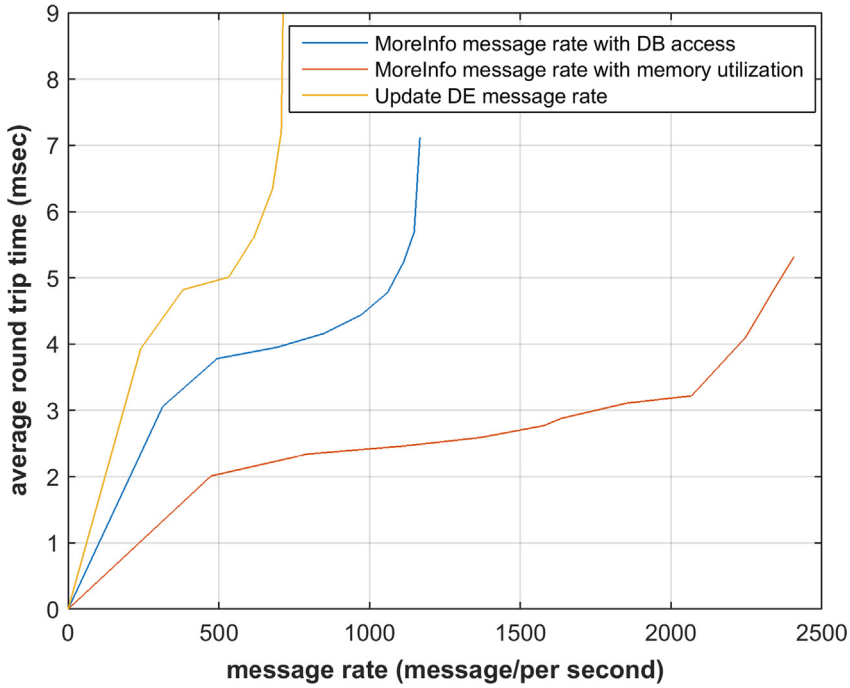

Fig. 12. Update DE and MoreInfo Message Rate with DB and Memory Access.

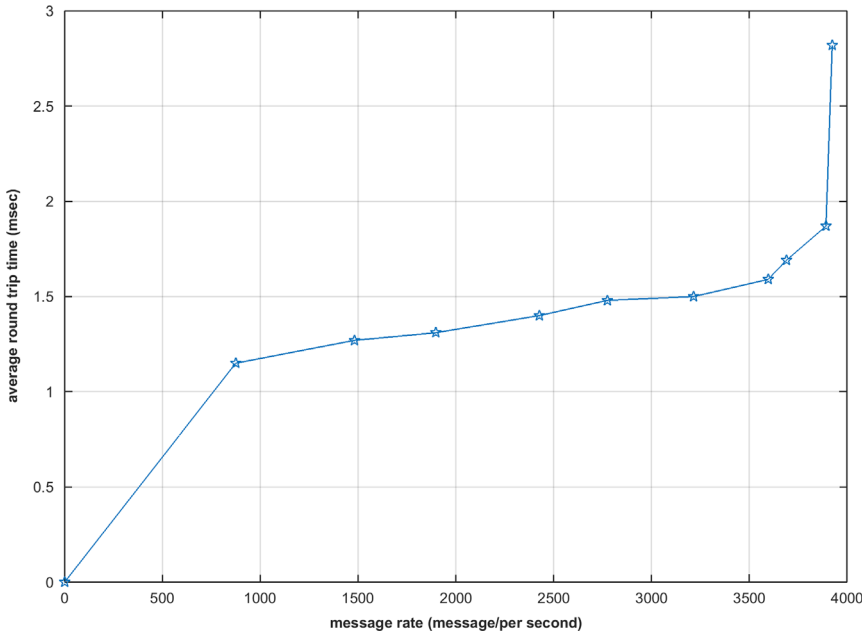

Fig. 13. Verification of the Service Message Rate.

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. The proposed system also plays a crucial role in protecting and keeping users' data synchronized with other sources. Furthermore, it makes data accessible for users when an integrated service retires and/or discontinues its services. Even though an analyzed service (Connotea) stopped its services in March, 2013, we have provided analysis results pertaining to that specific service for sake of comparison. Also, we think that our analyses would provide helpful pointers and results if the new versions of the service are made operational again in the future.

To achieve unification, the Digital Information Service is designed as a generic system with front and backend abstraction layers supporting one-to-many local information systems and their communication protocols. To achieve

federation, the Digital Information Service is designed to support information integration technique in which metadata from several heterogeneous sources are transferred into a global schema referred as *Merged Schema* and queried with a uniform query interface.

We performed a set of experiments to evaluate the performance and scalability of the prototype implementation of the Digital Information Service to understand whether it can achieve information federation with acceptable costs. This evaluation pointed out the following results. First, the Digital Information Service achieves information federation with negligible processing overheads for accessing/storing metadata. Second, the proposed study achieves noticeable performance improvements in standard operations by employing in-memory storage while preserving persistency of information. Third, the Digital Information Service scales to high message rates and message sizes while supporting information integration where metadata coming from different data storing systems.

With this research, we revisited distributed data management techniques to achieve integrated access to annotation metadata coming from a different number of annotation tools. We intend to further improve this approach to be able to scale up to a high number of distributed metadata sources such as video collaboration domain (YouTube etc.) and social networking domain (Facebook etc.). An additional area that we intend to research is an information security mechanism for the distributed Digital Information Service and machine learning techniques to identify typing errors within the documents.

Appendix A. Supporting information

Supplementary data associated with this article can be found in the online version at http://dx.doi.org/10.1016/j.is.2015.07.007.

References

[1] S.V. Adve, K. Gharachorloo, Shared memory consistency models: a tutorial, Computer 29 (12) (1996) 66–76.
[2] J.F. Allen, G. Ferguson, Actions and events in interval temporal logic, J. Log. Comput. 4 (5) (1994) 531–579.
[3] R. Alur, T.A. Henzinger, Reactive modules, Form. Methods Syst. Des. 15 (1) (1999) 7–48.
[4] P. Atzeni, F. Bugiotti, L. Rossi, Uniform access to NoSQL systems, Inf. Syst. 43 (2014) 117–133.
[5] J. Bates, J. Bacon, K. Moody, M. Spiteri, Using events for the scalable federation of heterogeneous components, in: Proceedings of the Eighth ACM SIGOPS European Workshop on Support for Composing Distributed Applications, ACM, 1998, pp. 58–65.
[6] H. Bulut, S. Pallickara, G. Fox, Implementing a NTP-based time service within a distributed middleware system, in: Proceedings of the Third International Symposium on Principles and Practice of Programming in Java, Trinity College Dublin, 2004, pp. 126–134.
[7] G. Coulouris, J. Dollimore, T. Kindberg, Distributed Systems: Concepts and Design, 3rd ed., Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001, ISBN:0-201-61918-0.
[8] K.R. Dittrich, S. Gatziu, Time issues in active database systems, in: Proceedings of the International Workshop on Infrastructure for Temporal Databases, Arlington, TX, 1993.
[9] G.C. Fox, Collaboration within an Event based Computing Paradigm. Retrieved January 23, 2015, from ⟨http://aspen.ucs.indiana.edu/collabtools/extras/indianafeb01_files/v3_document.htm⟩, 2001.
[10] G.C. Fox, S. Pallickara, A Scalable Durable Grid Event Service. Retrieved January 23, 2015, from ⟨http://surface.syr.edu/eecs/123/⟩, 2001.
[11] G. Fox, S. Pallickara, Deploying the naradabrokering substrate in aiding efficient web and grid service interactions, Proc. IEEE 93 (3) (2005) 564–577.
[12] S. Gatziu, K.R. Dittrich, Events in an Active Object-Oriented Database System, Springer, London, 1994, 23–39.
[13] P.S. Kam, A.W.C. Fu, Discovering Temporal Patterns for Interval-Based Events, Springer, Berlin Heidelberg, 2000, 317–326.
[14] T. Kindberg, G. Coulouris, J. Dollimore, J. Heikkinen, Sharing objects over the Internet: the Mushroom approach, in: Proceedings of the Global Telecommunications Conference, 1996. GLOBECOM'96. 'Communications: The Key to Global Prosperity, IEEE, 1996, November, pp. 67–71.
[15] R. Kowalski, F. Sadri, Towards a unified agent architecture that combines rationality with reactivity, in: Dino Pedreschi, Carlo Zaniolo (Eds.), Logic in Databases, Springer, 1996, pp. 135–149.
[16] H.T. Kung, J.T. Robinson, On optimistic methods for concurrency control, ACM Trans. Database Syst. 6 (2) (1981) 213–226.
[17] L. Lamport, Time, clocks, and the ordering of events in a distributed system, Commun. ACM 21 (7) (1978) 558–565.
[18] R. Li, D. Li, C. Sun, A time interval based consistency control algorithm for interactive groupware applications, in: Proceedings of the 10th International Conference on Parallel and Distributed Systems, ICPADS 2004, IEEE, 2004, pp. 429–436.
[19] C. Liebig, M. Cilia, A. Buchmann, Event composition in time-dependent distributed systems, in: Proceedings of the IFCIS International Conference on Cooperative Information Systems, CoopIS'99, IEEE, 1999, pp. 70–78.
[20] G. Liu, A.K. Mok, P. Konana, A unified approach for specifying timing constraints and composite events in active real-time database systems, in: Proceedings of the Fourth IEEE Real-Time Technology and Applications Symposium, 1998, pp. 199–208.
[21] G. Lodi, L. Aniello, G.A. Di Luna, R. Baldoni, An event-based platform for collaborative threats detection and monitoring, Inf. Syst. 39 (2014) 175–195.
[22] D. Mosberger, Memory consistency models, ACM SIGOPS Oper. Syst. Rev. 27 (1) (1993) 18–26.
[23] A.F. Mustacoglu, G.C. Fox, Performance of a collaborative framework for federating distributed digital entities, in: Proceedings of the International Symposium on Collaborative Technologies and Systems (CTS), IEEE, 2010, May, pp. 603–610.
[24] S.A. Myron, R. Knepper, M. Link, C. Stewart, PubsOnline: open source bibliography database, in: Proceedings of the 33rd Annual ACM SIGUCCS Fall Conference, 2005, pp. 247–249.
[25] S. Pallickara, G. Fox, NaradaBrokering: a distributed middleware framework and architecture for enabling durable peer-to-peer grids, in: Markus Endler, Douglas Schmidt (Eds.), Middleware, Springer, Berlin Heidelberg, 2003, pp. 41–61.
[26] P.R. Pietzuch, B. Shand, J. Bacon, A framework for event composition in distributed systems, in: Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware, Springer-Verlag New York, Inc. 2003, pp. 62–82.
[27] E. Rahm, P.A. Bernstein, A survey of approaches to automatic schema matching, VLDB J. 10 (4) (2001) 334–350.
[28] D.S. Rosenblum, B. Krishnamurthy, An event-based model of software configuration management, in: Proceedings of the Third International Workshop on Software Configuration Management, ACM, 1991, pp. 94–97.
[29] Y. Saito, M. Shapiro, Optimistic replication, ACM Comput. Surv. 37 (1) (2005) 42–81.
[30] M. Scott, R.P. Boardman, P.A. Reed, T. Austin, S.J. Johnston, K. Takeda, S.J. Cox, A framework for user driven data management, Inf. Syst. 42 (2014) 36–58.
[31] A.S. Tanenbaum, M. Van Steen, Distributed Systems: Principles and Paradigms, 2nd Ed., Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2002.
[32] R. Tolksdorf, Laura: a coordination language for open distributed systems, in: Proceedings of the 13th International Conference on Distributed Computing Systems, IEEE 1993, pp. 39–46.
[33] P. Wyckoff, S.W. McLaughry, T.J. Lehman, D.A. Ford, Tspaces, IBM Syst. J. 37 (1998) 454–474.