A Distributed Web Information Systems Platform Supporting High Responsiveness and Fault Tolerance *

Jordi Bataller, Hendrik Decker, Luis Irún and Francesc D. Muñoz

Instituto Tecnológico de Informática, Universidad Politécnica de Valencia
{bataller, hendrik, lirun, fmunyoz}@iti.upv.es

Abstract. Distributed replication of databases underlying web information systems is a viable way to solve problems of responsiveness and fault tolerance. We describe the middleware platform DIWISA for transparent object-oriented development of distributed web information systems. Support for distributed replication of information as well as fault tolerance and error recovery of web information systems is innovative: protocols which suit the given application best can be plugged in.

1 Introduction

A web information system (WIS) is an information system (IS) which may access sources of data via the world-wide web (shortly, *web*), for retrieving information queried by users.

A frequent nuisance with many WISs is their lack of responsiveness, the non-availability of underlying databases, slow-down of access, disturbance of transmission, delays of data delivery, downtimes of web servers, gateways and local networks, the temporary non-availability of stored information, and also recovery problems in case of breakdowns. Consistent restoration and refreshment of retrieved information usually involves re-transmissions of related data. This additional burden on the network load may slow-down WIS performance even more. Thus, measures for supporting the responsiveness, fault tolerance and availability of WISs are in order.

In general, a tried and tested principle for solving the problems of responsiveness, fault tolerance and availability is the distribution and replication of data. Hence, we are proposing to make use of this principle also in WISs, for making them more dependable. This is the idea behind the concept of a "distributed web information system", defined as follows.

A distributed web information system (DWIS) is a WIS with data sources that are replicated over several web sites, so as to speed up retrieval and updating, enable fault tolerance and enhance the availability of information and services on the web.

Clearly, distribution is not meant in the sense of mere partitions, but in the sense of a partial or complete replication of data. It is immaterial whether

* This work has been supported by the Spanish CICyT grant TIC2003-09420-C02-01.

the replica sites are physically close to each other (as in clusters or local area networks) or dislocated (as in wide area networks). In fact, the architecture to be introduced below is as general as to be able to operate in either fixed, mobile or hybrid networks.

A WIS has at least one (if not several or all) of the following three kinds of usages of the web as an interface:

- (a) between IS and user;
- (b) between search engine and data source;
- (c) between server nodes of dislocated data sources.

Point (a) entails that access to and communication with information sources may be remote. Hence, performance and availability may depend on the network's current load and the network nodes' current state, which is supposed to be alleviated by replicated distribution. As for point (b), issues related to the search of information on the web are not addressed here. Rather, we assume that the location of retrieved information needs not be searched but is is known to the WIS being queried. What matters with regard to point (c) is the web's internet protocol backbone which networks the replica of a DWIS.

In section 2, we further argue the case of enhancing WISs by underpinning them with replicated distribution. Subsequently, we introduce and discuss DIWISA, a distributed web information systems architecture for enabling high responsiveness and fault tolerance. Its main technical advantage over more conventional architectures is that it allows to plug in protocols that are best suited for DWISs, as opposed to other web services with different requirements. In general, DIWISA supports the consistency of replica in function of requirements for availability, as described in [18] [17] [13]. Section 3 outlines salient features of the overall architecture. Section 4 describes DIWISA as a WIS development platform. Section 5 focuses on the core management modules of DIWISA, i.e., the coordination of protocols for consistent replication and error recovery. After section 6, on related work, we conclude.

2 Why Distribution and Replication in WISs?

Since its recognition as a distinguished field of R & D in a collection of articles edited in [15], WIS is an area of rapidly increasing significance. However, many of the key points raised in the cited special edition of CACM still seem to have not yet been fully taken into account. In particular, we are referring to statements such as "WIS development parallels traditional system engineering", "WISs continue to face the same [...] challenges of traditional information systems" [15], "WISs are information systems first, and web systems second", "WIS development should use the same disciplined principles [...] required to build successful non-web information systems" [7]. In turn, most information systems strongly rely on transaction processing systems and database management technology, and WISs are no exception. Thus, it is justified to ask what database technology can contribute to the underpinning of WISs.

An issue of utmost importance for the success of a WIS is user acceptance. What counts most, beyond the surface of a user-friendly human-computer interface, are performance issues, such as the system's response time behaviour and the availability of information resources. Here, database technology has lots to offer, and not just in terms of indexing and tuning. The field of distributed databases has brought forward scalable solutions to improve responsiveness and fault tolerance, and thus availability, of persistently stored information [11] [21], also for web-based databases [20] [1].

This line of development has been further advanced in [17] [13] [5], where the middleware platform COPLA of the GlobData project [10] is discussed. In a follow-up project, one of the applications for which the middleware, now called DIWISA, is being re-configured and specialized, is web-based information systems. Besides the basic functionality of maintaining the consistency of replicated information, DIWISA brings several more advantages to WISs.

- DIWISA is a platform for supporting both the development and the operation of WISs, in terms of responsiveness and fault tolerance.
- Besides protocols for replication, DIWISA also hosts protocols for failure recovery, which further increases its capabilities to ensure responsiveness and fault tolerance.
- DIWISA supports concurrent WIS querying and concurrent conflict-free WIS updating.
- DIWISA offers a uniform object-oriented view on and access to information, even if underlying data structures are relational.
- DIWISA provides an IDL-based API with which it is possible to federate heterogeneous information sources into a uniform object-oriented format.
- DIWISA is equally well suited for both LANs and WANs, including wireless and mobile networks.
- DIWISA's orthogonal combination of WISs with distributed systems technology reconciles principles of client/server and web-based architectures (peer-to-peer, Grid, etc).
- In a DIWISA-based WIS, TCP/IP is multiply used for transmission of content as well as for broadcasting updates and synchronization messages to database replica. Hence, with suitable broadband technology (WLAN, DSL, GPRS, UMTS), all of querying, multimedia data transfer, retrieval and transaction processing as well as low-level signaling can be done via a single web link, which is particularly interesting for mobile users with small terminal devices such as notebooks, PDAs or even cellular phones.

Some of these issues are going to be addressed again in the remaining sections. More information on each of the points above can be found in publications from the GlobData project, as cited above.

3 DIWISA – Salient Features and Overall Architecture

The DIWISA middleware provides developers and users with a uniform object-oriented view of a singleton WIS, whose data sources however are replicated on a set of networked server nodes which use the web for communication and coordination. Replication is transparent to WIS services developed with DIWISA.

DIWISA supports consistency of replicated information across several web nodes by a multitude of protocols. They enable high responsiveness and transparent access to distributed information provided by WISs, also in case the web suffers from increased data traffic. Likewise, they also enable fault tolerance and continuity of operation in case of broken links or broken network nodes. DIWISA factors out and hides away the complications of distribution, so that web applications and services such as a WIS do not need any more complicated interfaces to underlying databases as traditional applications and service programs installed on a singleton server.

Neither WIS developers nor WIS users need to be aware of the distribution of WIS servers, which improves responsiveness and fault tolerance. In particular, the information system's underlying database and its scheme need to be defined and installed just once, instead of having to repeat that for each particular network node. Also, WIS querying and updating transactions as well as more involved applications can be developed and implemented on top of DIWISA as if they would operate on and interact with a single WIS, rather than a network of distributed system nodes.

3.1 The DIWISA Architecture

DIWISA is a middleware platform consisting of three layers, as depicted in fig. 1. This architecture is akin to the well-known three-layered structure of traditional information systems (application, logical and physical layer). The DIWISA top layer, called users layer, establishes the interface to developers and users. The middle layer, called core management layer, constitutes of the technical WIS core. The bottom layer, called storage access layer, provides the interface to stored information.

The users layer satisfies the purposes of WIS developers and programmers of applications on WISs on one hand, and those of end users of a DWIS supported by DIWISA, on the other. For developers and programmers, library functions are provided, for implementing WIS services. For end users, a web interface is provided. More details about the users layer are addressed in section 4.

The DIWISA core management layer consists of two major parts: a fixed and an exchangeable part. The fixed part serves queries and requests by users or WIS services and delegates tasks to the protocols module (which is the exchangeable part) and to the information store on the bottom layer. The protocols module is exchangeable in the sense that protocols for message passing, replication and consistency maintenance as well as failure recovery can be plugged into the core management layer. That way, the WIS administrator can always choose those protocols which are optimally suited for the given WIS service. Maintaining consistency of replicated information and restoring it in case the need of failure recovery arises, the protocols module caters for the coordination with the other core management modules by means of message passing over the web. More details related to the core management layer are addressed in section 5.

The storage access layer provides access to the information being queried or updated via the web interface on top. A local persistency device called *Uniform Data Store* (UDS) which maintains an object-oriented image of the underlying database is used as a private information store by each DIWISA core management module, i.e., for each replica. The UDS can be characterized as an abstraction of a general object database server, as it holds a copy of each replicated information object, which however might not always be up-to-date (the policy options for updating replicas for maintaining their consistency are going to be discussed in section 5). In fact, most of the translation and mapping between the data structures of the underlying information stores (which

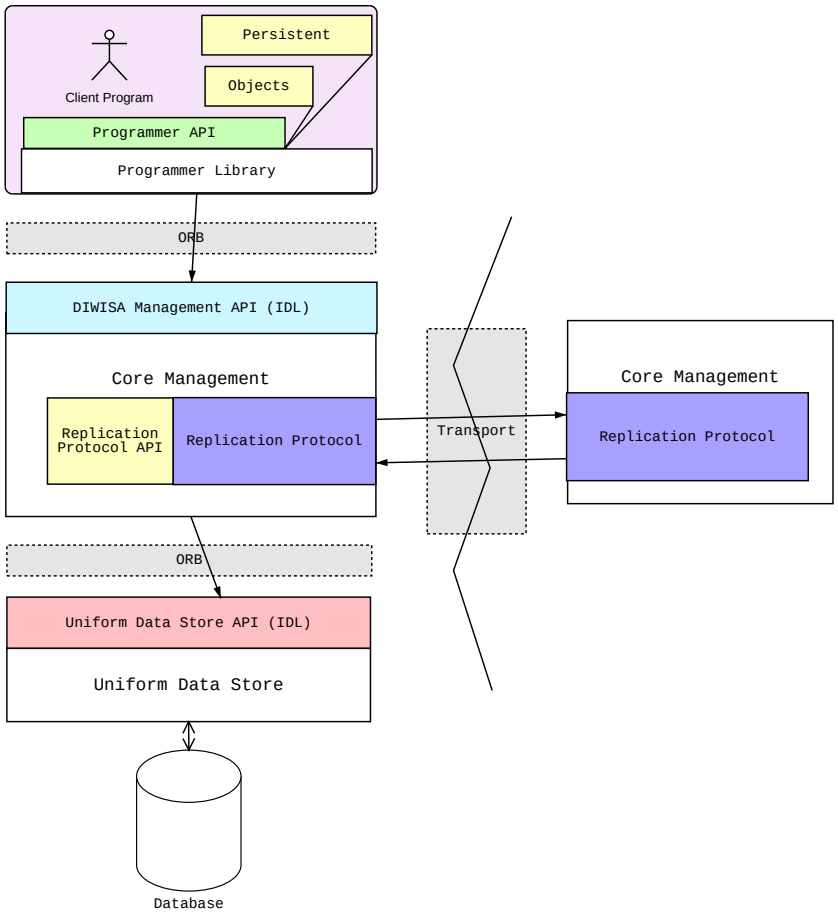

Fig. 1. DIWISA Architecture

may be a relational or an object-relational or an object-oriented database) and the uniform object-oriented access provided by the developers and users interface on top is accomplished by the UDS.

Interaction between the three DIWISA layers is enabled by a CORBA object request broker (ORB). The ORB mechanism yields several advantages:

1. The interface of each layer is specified in the OMG's interface definition language IDL. This ensures enhanced modularity as each layer may be implemented and upgraded independently in any convenient programming language, facilitating the versioning and maintenance of each module. In the released prototype, all three DIWISA layers and its interfaces have been implemented in Java, thus enabling platform independence of the entire system.

2. Also the routines in the DIWISA library on the users layer can be implemented in any programming language. That way, DIWISA can host WIS services developed in any language, on any platform with CORBA mappings.

3. Since CORBA interfaces are standard, the DIWISA communications system is robust against changes in the implementation of its ORB core. So, in case an improved ORB becomes available, it can be adopted for a new version of DIWISA without any code modification.

3.2 Object Orientation

The type of the actual database systems underlying the replicated ISs may be relational or object-oriented or object-relational, but that is immaterial to the WIS applications. Although DIWISA is not primarily meant as a tool for the federation of data sources or ISs, the underlying databases may very well be of heterogeneous type. In fact, interoperability of different kinds of databases underlying each replica node of a DWIS realized with DIWISA is feasible, provided that each database is mapped onto an equivalent UDS image. Our prototype works with PostgreSQL databases at each network node.

From the developers and end users point of view, information is uniformly modeled and accessed in DIWISA according to the object-oriented paradigm. Object orientation is particularly useful for a WIS because a lot of the information provided by WISs are either semi-structured and potentially complex objects or just large binaries, and hence not straightforwardly manageable in conventional databases of relational type.

By means of the object definition language GODL and the object query language GOQL, application programmers can adopt an uninhibited object-oriented point of view for developing queries, transactions and other WIS services. GODL and GOQL are simplified versions of the standard proposed by the Object Data Management Group [4]. Persistent information items (which actually are distributed data) appear both to the user and to the transaction control of the DIWISA core management (cf section 5) as if they were native objects (Java objects in our prototype implementation).

As already outlined above, another advantage is that developers and end users do not run the danger of getting lost in the intricacies and complications induced by distribution and replication, but rather can remain comfortable with a uniform object-oriented view on the DWIS as if it were a singleton WIS.

Implementation-wise, object orientation is enforced by proxies generated from the UDS schema definition. The GODL compiler generates a proxy class for each class defined in the UDS schema. Each proxy uses DIWISA to store or to retrieve the values of the information objects associated to it. This is transparent to (i.e., hidden from) developers and end users; WIS service applications can retrieve and update information by invoking appropriate methods in those proxies. Moreover, WIS services can be programmed to navigate through any chain of object relationships in the standard object-oriented manner, simply by stepping along the chain; at each step, another proxy is actually generated and instantiated with the corresponding object's values from the UDS.

Notice that this way of realising object orientation of an IS in DIWISA differs from the way object orientation is provided by object-oriented and object-relational DBMSs. There, the DBMS itself provides support for object management. However, our approach allows for greater flexibility and better support of the ODMG standard, in the sense that DIWISA can provide an object-oriented view even in case the underlying databases are purely relational.

4 The DIWISA Users Layer

The users layer interfaces to application programmers who develop and implement WIS services. End users and WIS application programs then deploy these services by means of methods provided via the API of a library module, which is a major component of the users layer. The library's methods communicate with DIWISA core managers for accessing persistent information, as if it were a singleton WIS. Retrieved information is presented to WIS services and users in the form of native objects. Since the users layer's interface is object-oriented, there is no need for WIS developers and users to be bothered with overcoming any impedance mismatch between the WIS service programs and the (relational or object-relational or object-oriented) structure of the underlying information store. What is even more important, from the architectural point of view, is that developers and users need not worry about replication, let alone the distributed locations of the IS server nodes.

In other words, the DIWISA users layer enables a replication-transparent view for WIS developers and end users to the searched information. More precisely, it provides the full range of location transparency, transaction transparency and fragmentation transparency. That is, neither programmers nor users need to know where or how the data are stored, which network communication protocol is used, and on which platforms the IS server nodes are operating. Moreover, failures in the network and information inconsistencies due to broken links or disconnected nodes will be seamlessly recovered.

The DIWISA users layer comprises three application programming interfaces for WIS service developers: the Management API, the Persistent Objects API and the Collections API. They enable a uniform object-oriented identification to retrieve, access and manipulate stored information objects, as well as sets, lists and bags thereof (see also 3.2). Technical details about these APIs and their implementation have been published in [8].

4.1 Optimising Responsiveness by Multithreading

It is possible to optimize WIS services by means of multiple threads. DIWISA supports different threads executing different WIS sessions and even multiple threads executed in the same session. For instance, several threads may be working on several collections, retrieving information objects in parallel. This feature is not well-handled by traditional information systems, as transactions are supposed to be serial streams of operations.

In order to coordinate threads on the same session, the users layer must know which threads are attached to each session. Opening a session immediately starts its first thread. Other threads willing to enter an ongoing session must call a particular registration method. Unregistering from a session is also an explicit operation. When a session is committed, aborted or closed by any thread, the rest of threads are notified about the event.

4.2 Optimising Responsiveness by Clever Caching

In principle, calls to information objects can be immediately forwarded from the users layer to the core management and storage layers. In that case, the calling thread is blocked until the operation has terminated its execution on the lower layers. However, this approach is not very efficient, in case many operations (possibly in the order of

thousands) are performed on the same object (e.g., when setting the attributes for a newly created complex object).

Here, an object cache implemented on the users level comes to the rescue, with two immediate benefits. Firstly, it avoids multiple retrievals of the same information in the same session. Secondly, calls to an information object are not immediately forwarded to the core management nor to the UDS. Instead, they are delayed, applied only in cache, and accumulated in a set of delayed operations. In case an action in a session depends on any of the delayed operations, the latter are jointly sent to the core management without further delay.

Another advantage of the cache is that it is shared by all currently running sessions, so that just one of them will actually retrieve an object initially from the storage layer. All of this is transparent to developers and users of WIS services. Thus, users may hardly notice any delay.

In general, UDS is accessed only when absolutely necessary. Actions are applied in blocks both by the core management layer and by the UDS. On the users layer, the amount of jobs in a set of delayed operations can be reduced before transmission whenever one of the operations supersedes the effect of any previous ones, which thus become superfluous. For instance, an object deletion supersedes any previous object modification. Even the whole set of delayed operations may be superseded, by being discarded in case the session is aborted.

The idea to minimise the delay caused by update operations by accessing cache memory instead of stored information hardly distinguishes DIWISA from common transaction handling in conventional database systems. However, to have it realised in a DWIS context which may host various replication strategies is noteworthy and in fact unprecedented. And there is even more potential for optimization: several consecutive updates on the same object may overwrite each other. Overwritten operations are exempted from the accumulated set of operations, thus avoiding redundant operations on the storage access layer. Finally, when accumulated operations are forwarded to the UDS, they are not applied sequentially (as they usually would, in a conventional database system). Instead, the DIWISA users layer groups and accumulates them per each object. Thus, only one call to the storage layer needs to be performed per object, which again avoids multiple storage access operations.

Performance tests runs have shown that the optimisations outlined above may dramatically reduce the amount of ORB communication and thus significantly improve WIS responsiveness and overall performance.

5 The DIWISA Core Management Layer

For each DWIS node, the DIWISA core management layer consists of two major manager modules: the core manager and the consistency protocol. The core manager is the fixed part, into which suitable protocols of choice can be plugged in, exchangeably. The core modules act as bridges to the modules on layers above, on the same level and below: users and WIS services, remote core managers on other nodes, and the local UDS. This scenario is illustrated in fig. 2. Essentially, the duties of these managers are, to

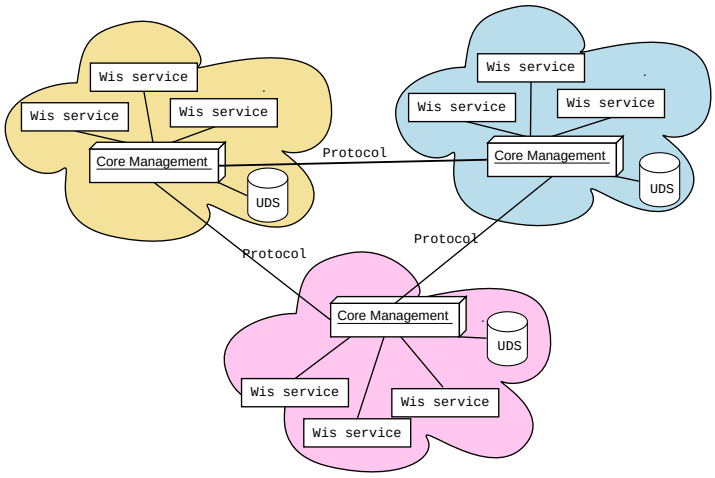

Fig. 2. Networked DIWISA Core Managers

- cater for requests by WIS services and end users,
- keep track of consistency of data locally cached in UDS,
- bring local data up to date as required by consistency policy,
- control sessions and decide when to commit,
- disseminate a session's updates to other managers,
- monitor liveliness of (links to) other managers,
- support recovery of failed or temporarily disconnected nodes.

The managers' actions are coordinated by a suitably chosen replication protocol. The protocols disposable for deployment in DIWISA are modular and exchangeable, i.e., can be suitably plugged in and out, thus allowing the WIS administrator to choose the most convenient of protocols for particular WIS domains and use cases. Hence, protocol modules which have been specifically tailored to the needs of a particular application can be selected for use at convenience. For instance, eager or lazy protocols, attuned to eager or lazy update policies (cf. [25] [9] [2]) can be used. Examples of available protocols in DIWISA are discussed in [18] [19] [17] [22] [14]. The appropriate choice of suitable protocols for a WIS depend entirely on specific WIS use cases (e.g., casual or business) and network properties (e.g., for mobile or fixed net users).

The core manager takes the role of a mediator in the system. First, it receives a WIS service request (through the execution of methods on the DIWISA users layer). Second, it informs the protocol about the request. Then, it waits for the protocol's orders, i.e., to execute a set of actions. Each such action locally addresses either a WIS service program or the UDS (cf. fig. 1).

Actions of core managers and protocols occur within WIS user sessions. Session management goes beyond what is usually offered by ISs, web-based or not. In fact, a DIWISA session is more general than a database transaction, or a sequence thereof, or even a set of concurrent transactions, since it may be in one of three consistency modes: *transaction*, *checkout* and *plain* mode.

– Transaction mode is the strongest mode. It guarantees that accessed information objects are always up-to-date. A successful commit of a session implies that modified objects are stored, thus defining new up-to-date versions. Transaction mode ensures atomicity, consistency, isolation and durability.

– Checkout mode is slightly weaker. Its consistency is similar to that of some tools for controlling the versioning of concurrently developed software, such as CVS [16]. Here, two sessions concurrently accessing the same information may conflict only if both try to modify it. Then, at least one of the session is aborted. In general, when two or more sessions are being checked to commit, the rules of the strongest mode apply.

– Plain mode sessions are only allowed to read. DIWISA ensures that all accessed objects belong to the same consistent view. This means that the version of each accessed object has been the most recent, at some point in the commit history.

Just as with the choice among disposable protocols, the mode of operation of each protocol can be determined at convenience, so as to optimally serve the needs suggested by a particular application, user preferences, network properties or other constraints.

6 Related Work

The mainstream of work in the field of WISs has thrived on the undeniable evidence that the web has profoundly influenced and changed notions associated to ISs. In particular, it has focused on the web as a provider of heterogeneous pieces of information and hypermedia data, and as a medium of visualisation and presentation of such information, with a strong (though not exclusive) emphasis on user interfaces (e.g., [23] [12]). As opposed to that, DIWISA approaches WISs from the traditional information systems' side, with a strong emphasis on engineering the availability of underlying databases on the level of their physical storage, i.e. a sector below the conceptual layers which take care of transforming data for preparing their representation on screen.

6.1 DIWISA's pedigree

DIWISA is an outgrowth of the GlobData project [10], which brought forward the middleware platform COPLA [8] [13]. COPLA supports the plugging in and out of appropriate replication protocols for realising convenient strategies of maintaining the consistency of replicated data. The use of this flexibility for increasing the availability of information has been pondered in [18] [17] [5]. DIWISA is still in the initial phase of an ongoing GlobData follow-up project. The main distinguishing points which set it apart from COPLA consists in the use of lazy protocols and the deployment of message recuperation for failure recovery, as opposed to eager replication which was favoured in COPLA. Lazy, i.e., optimistic strategies of replication are more viable in information systems (web-based or not) than in online transaction processing systems, since by far the most frequent kind of transactions will consist in read operations, while OLTP database systems which had been the main focus of COPLA usually require eager, pessimistic strategies. However, the tendentially higher volatility of web-based information on one side, and higher volumes of web data, on the other, as well as the streaming of web data, will certainly entail further tunings of protocols that are currently benchmarked in our prototype implementation.

6.2 Other Work

The issue of replicating databases over distributed web sites is not new. For instance, in [20], technologies to build distributed web hosting solutions are outlined. Northrup points out advantages of wide-area web-based replication of databases, suggesting to use them as a lower file management layer for simplifying the implementation of upper tiers. However, without a middleware support such as provided by DIWISA, the development of solutions as proposed in [20] tend to become exceedingly complicated and prohibitively costly.

A particular implementation of "distributed database versioning" is discussed in [1]. The described technique ensures strong consistency (1-copy serialisability) by an eager replication policy, while at the same time ensuring sufficient scalability, by means of a deadlock-free concurrency control algorithm based on a conflict-aware scheduler. The overhead of using 1-copy serialisability, as compared to less stringent consistency models based on lazy replication, is discussed. However, the proposed solution focuses on local area clusters and does not take wide area distribution into account. Moreover, it may suffer from shortcomings related to node failure management and inconvenient transmission requirements.

Also for local area clusters, the Neptune [24] as well as the TACT project [26] address the issues of availability and replication support for building web service infrastructures. Similar to DIWISA, they propose different levels or tunings of replication consistency in order to cater for varying consistency requirements of different web content applications. However, the considered requirements are motivated not just by availability, responsiveness and fault tolerance, but to a large extent also by load balancing, which however is out of the scope of DIWISA.

Similar to TACT, the Bayou system [6] presents a general framework for measuring consistency degradation in terms of user perception. In Bayou, consistency requirements are considerably relaxed due to the mobility and hence the occasional disconnectedness of network users. That allows Bayou replicas to be more lightweight than normal DBMS servers, however at the cost of being not useful for stronger replication requirements.

Common to all proposals mentioned in this section is their lack of generality, as compared to DIWISA. The latter provides a range of pluggable protocols such that the most suited one for a given WIS application can be chosen, as opposed to a fixed eager or lazy replication policy. Moreover, DIWISA supports both local and wide area networks of replica. And, last but not least, our middleware is fully implemented, it works and is continuously improved by austere benchmarking, while many proposals in the literature remain on the conceptual level.

7 Conclusion

We have discussed the DIWISA architecture. DIWISA is a middleware platform that provides a transparent object-oriented view for designing, developing and deploying distributed web information systems. A WIS developed with DIWISA can be replicated and distributed seamlessly over several dislocated sites, in order to enable fast responses and fault tolerance. The architecture is based on the simple idea that advanced web data engineering technology and traditional database management system principles can fruitfully be combined orthogonally in many ways, thereby underpinning a further convergence of both fields, yielding more reliable WISs with solid capabilities.

DIWISA boosts increased levels of responsiveness and fault tolerance. Both are enabled by a fully transparent TCP/IP-based distribution and replication strategy and a range of new protocols. The protocols are characterized by different combinations of properties such as eager or lazy update propagation, optimistic or pessimistic concurrency control, primary-copy or update-everywhere replicas etc, and various fine-tunings thereof. Each of these protocols can be plugged into the middleware's core management layer in order to cater suitably for the consistency and recuperation of replicated information. It is the application, the network load, device properties and user preferences which determine an optimal choice of appropriate protocols.

Ongoing investigations are concerned with the following issues.

- Improvements of existing protocols and development of new protocols which would serve best the particular needs of WISs and applications thereof.
- Rather than e-commerce applications, we intend to focus on public access WISs for distributed organizations such as e-government, public transport networks, Red Cross and Red Crescent, World Health Organization, etc.
- Reduction of the overhead of enforcing a pervasive object-oriented view on information items by re-implementations of large parts of the users layer and the storage access layer, adopting an object-relational point of view and using JDBC-based interfaces.
- Transformation and rendering problems of scaling and adapting retrieved information to the particular needs of users and capabilities of used platforms and devices are not dealt with in DIWISA. For mobile users and platforms, related solutions have been proposed, e.g., in [3]. We intend to look into combining such solutions with the DIWISA platform.

References

1. C. Amza, A. Cox, W. Zwaenepoel. Distributed Versioning: Consistent Replication for Scaling Back-end Databases of Dynamic Content Web Sites. *Proc. Middleware 2003*, Springer LNCS Vol. 2672, 282–304, 2003.
2. Y. Breitbart, R. Komondoor, R. Rastogi, S. Seshadri, A. Silberschatz. Update propagation protocols for replicated databases. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 28(2):97–108, 1999.
3. D. Carrega, H. Muyal, H. Decker, M. Wallbaum. Integrating voice and data services for mobile internet collaboration with the MOVE middleware architecture. *Proc. 1st Int'l Workshop on Web Based Collaboration*, 12th DEXA, IEEE Computer Society, 532–536, 2001.
4. R. Cattell, D. Barry, M. Berler, J. Eastman, D. Jordan, C. Russell, O. Schadow, T. Stanienda, F. Velez, editors. *The Object Data Standard: ODMG 3.0.* Morgan Kaufmann Publishers, January 2000.
5. H. Decker, F. Muñoz, L. Irún, P. Castro, A. Calero, J. Esparza, J. Bataller, P. Galdámez, J. Bernabéu. Enhancing the Availability of Networked Database Services by Replication and Consistency Maintenance. *Proc. Parallel and Distributed Databases*, 14th DEXA Workshop, IEEE Computer Society, 531–535, 2003.
6. A. Demers, K. Petersen, M. Spreitzer, D. Terry, M. Theimer, B Welch. The Bayou architecture: Support for data sharing among mobile users. *Proc. IEEE Workshop on Mobile Computing Systems & Applications*, IEEE Computer Society, 2–7, 1994.

7. A. Dennis. Lessons from Three Years of Web Development. *Comm. ACM* Vol. 41, No. 7, 112–113, 1998.
8. J. Esparza, A. Caleiro, J. Bataller, F. Muñoz, H. Decker, J. Bernabéu. COPLA - a middleware for distributed databases. *Proc. 3rd Asian Workshop on Programming Languages and Systems*, 102–113, 2002.
9. F. Ferrandina, T. Meyer, R. Zicari. Implementing Lazy Database Updates for an Object Database System. *Proc. 20th Int'l Conf. Very Large Databases*, 261–272, 1994.
10. GlobData website, http://globdata.iti.es, 2002.
11. R. Guerraoui, A. Schiper. Fault-Tolerance by Replication in Distributed Systems. *Proc. Int'l Conf. Reliable Software Technologies*, Springer LNCS Vol. 1088, 38–57, 1996.
12. G.-J. Houben, P. Barna, F. Frasincar, R. Vdovjak. Hera: Development of Semantic Web Information Systems. *Proc. 3rd Int'l Conf. Web Engineering*, Springer LNCS Vol. 2722, 529–538, 2003.
13. L. Irún, F. Muñoz, H. Decker, J. Bernabéu. COPLA: a platform for eager and lazy replication in networked databases. *Proc. ICEIS '03, Vol. 1*, 273–278, 2003.
14. L. Irún, F. Muñoz, J. Bernabéu. An Improved Optimistic and Fault-Tolerant Replication Protocol. *Proc. 3rd Int'l Workshop on Databases in Networked Information Systems*, Springer LNCS Vol. 2822, 188–200, 2003.
15. T. Isakowitz, M. Bieber, F. Vitali. Web Information Systems – Introduction. *Comm. ACM* Vol. 41, No. 7, 78–80, 1998.
16. R. Krause. CVS: An introduction. *Linux Journal* Vol. 87, 72–76, 2001.
17. F. Muñoz, L. Irún, P. Galdámez, H. Decker, J. Bernabéu, J. Bataller, M. Bañuls. Flexible management of consistency and availability of networked data replications. *Proc. FQAS*, Springer LNCS Vol. 2522, 289–300, 2002.
18. F. Muñoz, L. Irún, P. Galdámez, H. Decker, J. Bernabéu, J. Bataller, M. Bañuls. Globdata: A platform for supporting multiple consistency modes. *Proc. IASTED Int'l Conf. Information Systems and Databases*, 2002.
19. F. Muñoz, L. Irún, P. Galdámez, J. Bernabéu, J. Bataller, M.C. Bañuls. GlobData: Consistency protocols for replicated databases. *Proc. IEEE-YUFORIC'2001*, 97–104, 2001.
20. T. Northrup. Designing geographically distributed Web hosting solutions. *Win2000 Experts Journal* Vol. 2, No. 1, 19–25, 2001.
21. T. Özsu, P. Valduriez. Principles of Distributed Database Systems, 2nd Edition. Prentice Hall, 1999.
22. L. Rodrigues, H. Miranda, R. Almeida, J. Martins, P. Vicente. Strong replication in the Globdata middleware. *Proc. Workshop on Dependable Middleware-Based Systems* at Dependable Systems and Networks Conference, DSN, 2002.
23. A. Scharl. A conceptual, user-centric approach to modeling web information systems. *Electronic Proc. AUSWEB*, http://ausweb.scu.edu.au/aw99/papers/, 1999.
24. K. Shen, T. Yang, L. Chu, J. Holliday, D. Kuschner, H. Zhu. Neptune: Scalable Replication Management and Programming Support for Cluster-based Network Services. *Proc. 3rd USENIX Sympos. Internet Technologies and Systems*, ACM Press, 197–208, 2001.
25. M. Wiesmann, A. Schiper, F. Pedone, B. Kemme, G. Alonso. Database replication techniques: A three parameter classification. *Proc. 19th IEEE Sympos. Reliable Distributed Systems*, 206–217, 2000.
26. H. Yu, A. Vahdat. Design and Evaluation of a Continuous Consistency Model for Replicated Services. *Proc. 4th Sympos. Operating Systems Design and Implementation*, USENIX Assoc., 305–318, 2000.