What Service Replication Middleware Can Learn from Object Replication Middleware

Johannes Osrael
j.osrael@tuwien.ac.at

Lorenz Froihofer
l.froihofer@tuwien.ac.at

Karl M. Goeschka
k.goeschka@tuwien.ac.at

Institute of Information Systems, Vienna University of Technology
Argentinierstrasse 8, 1040 Vienna, Austria

ABSTRACT

Replication is a well-known technique to enhance dependability and performance in distributed systems. A plethora of replication middleware for distributed object systems has been proposed in the past decade. However, replication in service-oriented systems is still in its infancy. In this paper, we analyze some of the proposed service replication middleware solutions and compare them on an architectural level with object replication middleware. In particular, we focus on replication middleware that allows for (but is not limited to) strict consistency of replicas since this is required by many real-life applications. We identify six major infrastructure components and present a generalized architecture for both distributed object and service-oriented replication middleware. The result of our comparison is unambiguous: Replication middleware for service-oriented systems and distributed object systems (such as FT-CORBA) share many commonalities and only subtle differences caused by the different granularity of the replicated entity, or different transaction models.

Categories and Subject Descriptors

C.2.4 [**Computer - Communication Networks**]: Distributed Systems; D.2.11 [**Software Engineering**]: Software Architectures

General Terms

Design

Keywords

Replication, middleware, architecture, service-oriented systems, distributed object systems

1. INTRODUCTION

Service-oriented architectures are more and more adopted by industry in almost all areas of computing. If the success continues and the service-oriented computing approach is applied in critical, vital systems—such as air traffic control systems, health care systems, nuclear power plants, chemical refineries, public transportation, etc.—*dependability* needs to be ensured.

Dependability is the "ability of a system to avoid service failures that are more frequent or more severe than is acceptable" [4]. Fault tolerance, which ensures that a service failure can be avoided if faults are present in the system, is one of the techniques to achieve dependability.

Redundancy is the key to fault tolerance. Fault tolerance cannot be achieved without redundancy. *Replication* of both hardware and software resources is one important means to introduce redundancy and thus to enable fault tolerance. As pointed out by Ken Birman [8], "only replication can ensure access to critical data in the event of a fault."

Replication in distributed object, database systems, and file systems is well-established; however, only few replication solutions exist for service-oriented systems.

While replication of stateless services is comparatively easy to achieve, replication of stateful services requires synchronization of the replicas' state. A stateful service keeps state either in memory (non-persisted, transient state) or persists it in a data store (persisted state) such as a file or a database. The latter type of stateful service can be modeled as a stateless "access" service plus a stateful resource. Thus, for this type of service, state synchronization of replicas can be performed either via the underlying data store (i.e., a database or file replication system) or via the service interface. The few service replication middleware solutions proposed so far are not restricted to services that store their state in a data store and thus perform replication on the service level and not on the data level. Unfortunately, some of the proposed service replication middleware systems are obviously not fault tolerant (e.g., [15]) since many subtleties (e.g., crash of a primary replica during update propagation to backups, etc.) are not yet considered. Thus, we believe closer cooperation between the dependability and the service-oriented community is needed to reach sufficient maturity, necessary for deployment of service-oriented systems in safety- and mission-critical settings.

Thus, in this paper we contribute to the understanding of replication middleware by an architectural comparison of some of the rather mature and well-designed service-oriented solutions with well-established replication middleware for distributed object systems. In particular, we focus on middleware that provides replication protocols that allow for strict replica consistency (e.g., linearizability [3]) since this is required by many real-life applications. For example, primary-backup [9] replication (also known as passive replication) and active [25] replication allow[1] for strict consistency. In the former variant, the primary replica processes the request and forwards the update to the backup replicas. In the latter variant, the client sends the request to all replicas—either directly or via a communication module [27].

We consider middleware that can cope with crash failures [11] and link failures but we do not focus on Byzantine fault-tolerant middleware (e.g., [17]) since Byzantine behavior is rarely encountered in practice.

Paper Overview. Section 2 presents state-of-the art middleware architectures, both from the service-oriented and distributed object world. Based on these representative architectures, six major infrastructure components for replication middleware are identified in Sect. 3. Furthermore, the commonalities and differences of these building blocks with respect to service-oriented and distributed object systems are discussed. Section 4 covers related work before we draw our conclusions in Sect. 5.

2. MIDDLEWARE ARCHITECTURES

Few middleware solutions have been proposed for service-oriented systems: The middleware systems presented in [29, 23] offer transparent active replication based on group communication [10]. Primary-backup replication of Web services is offered by the FT-SOAP (Fault-Tolerant SOAP) middleware [16]. Thema, a byzantine fault-tolerant middleware for Web service applications is proposed in [17]. ADAPT [5] is a J2EE replication framework integrated into the JBOSS application server that allows to plug-in replication protocols. Besides replication of Enterprise JavaBeans it supports replication of AXIS Web services as well. The Web services might contain session state but services that invoke other EJBs or call a database are not supported. Among these few solutions, we concentrate on FT-SOAP [16] and the active replication middleware systems [29, 23] since primary-backup and active replication are the most commonly used replication techniques in real-world applications. We compare these middleware systems with the FT-CORBA (Fault-Tolerant Common Object Request Broker Architecture) specification [19] and the DeDiSys replication middleware [20] for distributed objects. We have chosen FT-CORBA since it is a well-known standard and DeDiSys since it has been implemented on the three major middleware platforms J2EE, Microsoft .NET [21], and CORBA [7]. FT-CORBA supports both active and passive replication while DeDiSys is targeted to the latter replication model.

2.1 Service-Oriented Systems

2.1.1 Primary-Backup Replication Middleware

Figure 1 shows the architecture of the FT-SOAP replication middleware [16]. The original presentation of the architecture in [16] has been redrawn to allow for easier comparison with the object replication middleware. As in the FT-CORBA architecture, the key components in FT-SOAP are the *Replication Manager*, the *Fault Management* unit and the *Logging & Recovery* unit. SOAP engine interceptors are used to intercept client requests. WSDL files of replicated services are published in a UDDI registry.

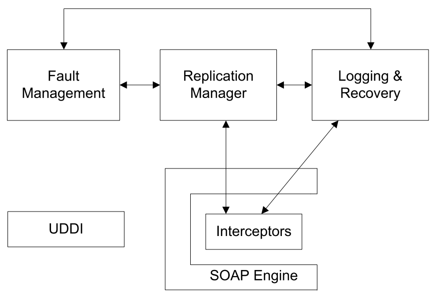

Figure 1: FT-SOAP

The FT-SOAP replication manager basically provides interfaces for (i) setting the desired replication properties like the replication style, initial number of replicas, etc. and (ii) creating and managing service groups. The fault management unit, used to monitor replicated services, consists of a fault detector and a fault notifier. In FT-SOAP, a centralized reliable file system is used to log invocations in order to allow for recovery processes. If necessary, invocations are replayed during the recovery stage.

2.1.2 Active Replication Middleware

The middleware presented in [29] offers transparent active replication based on group communication. The system implements the TOPBCAST [14] probabilistic multicast protocol using the JGroups toolkit [1]. Both synchronous and asynchronous interaction between the client and the Web service are supported.

WS-Replication [23] is a framework for wide area replication of Web services and offers transparent active replication. The major components in WS-Replication are a *Web service replication component* and a *reliable multicast component*. The former component enables active replication of Web services while the latter—called WS-Multicast—provides SOAP-based group communication. Moreover, WS-Multicast performs failure detection (which is required for group communication) by a SOAP-based ping mechanism. WS-Multicast can also be used independently from the overall WS-Replication framework for reliable multicast in a Web service environment. The SOAP group communication support has been built upon the JGroups [1] toolkit.

2.2 Distributed Object Systems

2.2.1 Fault-Tolerant CORBA Architecture

Originally, CORBA [19], a popular middleware framework for object-oriented distributed systems, lacked of support for fault tolerance. Thus, FT-CORBA [19] has been introduced to overcome this short-coming. The FT-CORBA standard defines an architecture which supports a range of fault tolerance strategies, including active and passive replication of CORBA objects. In FT-CORBA, replicated objects constitute an object group, which is referenced by an Interoperable Object Group Reference (IOGR). Clients invoke operations on object groups. The core components of the FT-CORBA

[1]Of course, primary-backup replication can also be configured for weaker forms of consistency, e.g., by employing a lazy update propagation scheme.

infrastructure are the *Replication Manager*, the *Fault Management* unit, and the *Logging & Recovery* unit. The specification defines replication manager interfaces for operations to (i) set replication properties (e.g., passive vs. active replication), (ii) manage object groups (e.g., add/remove members), and to (iii) create and destroy objects. A fault detector monitors replicated objects, servers, or processes and reports faults to a notification component, which in turn propagates the information to interested components, e.g., the replication manager. This allows the replication manager to act upon reported faults, e.g., to choose a new primary if the original fails, to create new replicas, etc. Logs, containing the state and actions, are maintained per object group by the logging mechanism. The recovery mechanism processes the log and brings new, recovering, or backup replicas to the current state.

FT-CORBA does not specify group communication primitives. Thus, in order to provide for example active replication, FT-CORBA implementations must use proprietary group communication mechanisms [6].

2.2.2 DeDiSys Replication Architecture

Figure 2 shows the replication architecture of the platform-independent DeDiSys middleware [20]. DeDiSys builds upon standard middleware like J2EE, Microsoft .NET, or CORBA. Thus, some of the DeDiSys components are already provided by the standard middleware, e.g., the Transaction Service. Furthermore, DeDiSys uses off-the-shelf stable storage mechanisms (typically a database) and existing persistence solutions (e.g., persistence frameworks).

The core components of the DeDiSys replication architecture are the *Replication Manager*, the *Replication Protocol*), the *Group Membership Service*, and the *Group Communication* component. Further components are the *Invocation Service*, the *Naming Service*, the *Transaction Manager*, the *Activation Service*, and the *Persistence/Stable Storage* component.

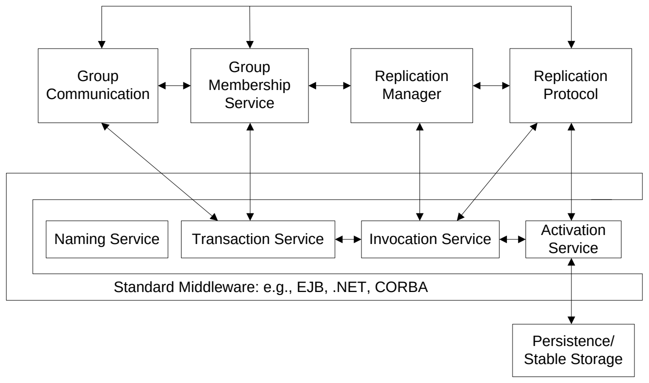

Figure 2: DeDiSys Replication Middleware

The *Replication Manager* keeps track of object replicas in the system. Thus, it maintains a mapping between global object IDs and replica IDs with their location and role (primary or backup replica). The DeDiSys replication manager supports the passive replication model. A location service for replicas is closely integrated with the replication manager.

The *Replication Protocol* component provides the specific replication logic for a given replication protocol. In case of primary-backup replication, client invocations are initially processed at the primary replica. Afterwards, the updates are propagated to the backup replicas. Besides the replication logic during healthy periods, the behavior of the protocol in the presence of faults needs to be specified in this component. Furthermore, synchronization with other replicas in case of the recovery of a replica needs to be performed by the replication protocol. In addition, some replication protocols, like the primary-per-partition protocol [20] require logging of operations or states.

A *group membership service* is used to keep track of which nodes are operational, taking into account intentional group changes (join or leave) as well as node and link failures. *Group communication* provides reliable multicast to groups with configurable delivery and ordering guarantees.

Group membership and group communication services can be treated as separate components which interact with each other. However, in practice, both components are usually integrated in one toolkit which is referred as view-oriented group communication system [10].

3. ARCHITECTURAL COMMONALITIES AND DIFFERENCES

3.1 Generalized Architecture

Based on the representative architectures discussed in Sect. 2, six major architectural units for both object and service replication middleware can be identified as shown in Fig. 3: A *Multicast Service*, a *Monitoring Service*, a *Replication Manager*, a *Replication Protocol* unit, an *Invocation Service* and an optional *Transaction Service*. Some of the units such as the replication protocol and the multicast service are naturally distributed since they realize distributed algorithms. The other components should also be implemented in a distributed fashion in order to avoid single points of failure and to provide an adequate level of fault-tolerance for the infrastructure itself. For instance, a replication manager instance resides on every node in the system that hosts business services/objects. Even more so, the state of the replication manager is also subject to replication. Besides these six major components, replication middleware typically comprises further supportive components such as a naming service (e.g., for resolving human-readable names to identities) or some kind of persistence service (e.g., for object-relational mapping).

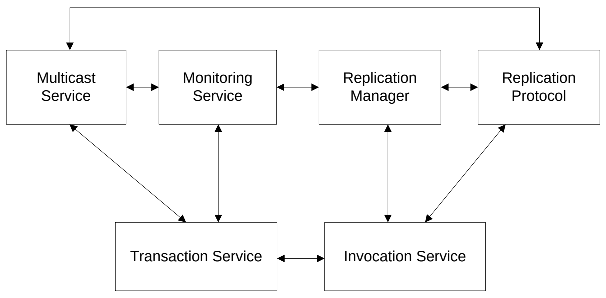

Figure 3: Generalized Replication Architecture

The infrastructure components share many conceptual commonalities and have only subtle differences with respect to their realization in service or object replication middleware. We discuss them in detail in the next subsections.

3.2 Multicast Service

Reliable multicast primitives are needed both in object and service replication middleware, e.g., for propagation of updates from the primary to the backup replicas in case of passive replication. Schiper [24] points out that group communication, a communication infrastructure that eases the implementation of replication techniques, is beneficial for both active and passive replication. Active replication requires ordering of operations, which can already be provided by a group communication primitive. Group communication primitives hide most of the (implementation) complexity of passive replication: For instance, group communication allows to cope with undesirable situations such as the crash of the primary during a multicast. Whether the replicated entity is a service or an object does not impose any conceptual differences in this respect. Indeed, most of the presented middleware architectures rely on group communication. Examples for state-of-the-art group communication toolkits are JGroups [1], Spread [28], or the newly proposed SOAP-based WS-Multicast toolkit [23], which is specifically targeted to service-oriented systems. The only noteworthy difference compared to traditional group communication toolkits is that WS-Multicast exposes its operations via a WSDL (Web Services Description Language) [26] interface. However, this could also be realized for the other toolkits.

3.3 Monitoring Service

Monitoring of the replicated entities and detection of faults is required both in distributed object and service-oriented systems since replication middleware needs to take appropriate actions in case of a fault. For instance, in case of passive replication, it has to promote a backup to a new primary replica if the original primary crashes. If a group communication toolkit is used as in DeDiSys [20] and WS-Replication [23], the group membership unit already provides this functionality on the node level.

3.4 Replication Manager

Both in distributed object and service-oriented systems, some component is necessary which manages replicated services/objects, including tasks such as storing the location and role of replicas, maintaining service/object groups, general configuration of the replication middleware such as the replication style, etc. Typically, this component is called the replication manager (e.g., in FT-SOAP [16], DeDiSys [20], FT-CORBA [19]). The *Web service replication component* of the WS-Replication framework [23] provides similar functionality. Though the tasks of this component are identical for service and object replication middleware, minor differences are caused by the different granularity (typically coarse-grained in case of services and usually fine-grained in case of objects) and the number of the replicated services/objects. For instance, a replication manager in object replication middleware typically needs to maintain a huge number of objects (e.g., millions) while the number of services that need to be replicated is comparatively small. This, for example, influences the choice of the data structure for the replica location service which is part of the replication manager.

3.5 Replication Protocol

The actual replication logic (i.e., triggering of update propagation, recovery, etc.) is typically either a separate component (e.g., in the DeDiSys replication middleware) or embedded in the replication management unit (e.g., in the CORBA-based Eternal system [18] or in the WS-Replication middleware [23]). The advantage of a separate replication protocol component is that a change of the protocol (e.g., necessary due to system evolvement) is easier to achieve. Again, in this respect no differences between service-oriented and distributed object middleware arise.

3.6 Invocation Service

An invocation service provides the invocation logic used for invocation of operations and provides specific guarantees with respect to node or link failures. It further provides the possibility to intercept service/object invocations and transmits additional data with an invocation, e.g., the identifier of a transaction to associate a specific call with a transaction. Both distributed object replication middleware and service replication middleware exhibit such interceptors: For instance, the DeDiSys replication middleware [20] uses the interception mechanism of the JBOSS application server (in the J2EE version of the framework) and .NET Remoting interceptors (in the .NET variant). The Web service replication component of the WS-Replication framework [23] comprises a proxy generator which generates a proxy for each Web service operation. The proxy intercepts invocations to the replicated Web service and triggers the further replication logic.

3.7 Transaction Service

Transactions are a fault tolerance technique by themselves; specifically the atomicity and durability properties of traditional ACID transactions are related to fault tolerance [24]. Atomicity denotes that either all or none of the transaction's operations are performed. Durability requires that the committed effect of transactions is permanent, such that the data are available after a failure or system restart. However, since durability has its limitations (e.g., some failures such as a disk crash are not recoverable), replication needs to be introduced in critical transactional systems. Thus, transactions need to be performed on replicated objects or services. While distributed object replication middleware often supports transactions (e.g., DeDiSys middleware [20]), support for transactions in replication middleware for service-oriented systems is rather in its infancy. Up to our knowledge, only WS-Replication [23] has been combined with transactional support. WS-Replication has been successfully tested in combination with long running transactions as defined in the Web Services Composite Application Framework (WS-CAF) [2]. In contrast to short-running ACID transactions, long running transactions provide atomicity guarantees despite relaxing the isolation property. Compensation actions are required to reverse the effects of long running transactions in case of conflicts. Although the combination of WS-Replication with transactions yielded promising results, there is clearly a need for further research in this area, especially with different replication protocols and other transaction models.

Table 3.7 summarizes the commonalities and differences of object and service replication middleware.

	Object Replication Middleware	Service Replication Middleware
Granularity	typically fine-grained (objects)	typically coarse-grained (services)
Multicast Service	group communication beneficial	group communication beneficial
Monitoring Service	failure detector, group membership service	failure detector, group membership service
Transaction Service	typically ACID transactions	ACID and long running transactions
Replication Manager	maintains large number of replicated entities	maintains small number of replicated entities
Replication Protocol	separate or embedded	separate or embedded
Invocation Service	interceptors	interceptors

Table 1: Commonalities and Differences

4. RELATED WORK

Up to our knowledge, no comparison of object and service replication middleware architectures exists in literature. Similarities between these two worlds with respect to replication are mentioned in [23] but not described. Although object replication middleware is well-established, architectural comparisons are even hard to find within this area. One noteworthy exception [12] discusses both early fault tolerant CORBA implementations and the FT-CORBA standard; however, with a strong focus on the latter and no relation to service-oriented systems. Besides this excellent CORBA-specific work, "related work" sections of replication middleware papers (e.g., [22]) typically contain brief comparisons, which lack in-depth coverage as our comparison provides. Moreover, most of the scientific papers about replication in the traditional distributed computing field focus on the algorithms and not on the middleware providing the replication protocols. Thus, the replication architecture is often not described at all or rather implicitly. Fortunately, as seen in Sect. 2, at least some of the replication middleware architectures in both the distributed object and service-oriented field are well described and allow a thorough comparison.

Replication of data stores such as databases or file systems, which can be used on the data level of a service-oriented system if stateful services persist their state in a data store, has been extensively discussed in scientific literature. Oliveira et al. give an excellent overview about state-of-the-art in database replication [13]. Wiesmann et al. [27] compare replication protocols in database systems with replication in distributed object or process systems but do not focus on middleware architectures.

5. CONCLUSIONS

In this paper, we compared state-of-the-art service replication middleware with object replication middleware on an architectural level. Since strict consistency is required by many real world applications, the focus of our comparison was on frameworks that provide the most common replication techniques that allow for strict consistency, namely active and passive replication.

Replication is rather in its infancy in service-oriented systems; thus, only a small number of replication middleware solutions was suitable for our comparison [23, 16, 29].

Among the replication architectures for distributed object systems we have chosen the well-known FT-CORBA standard [19] and the DeDiSys architecture [20] due to its realization on the three major middleware platforms J2EE, Microsoft .NET, and CORBA.

The result of our comparison is unambiguous: Object and service replication middleware share many commonalities and only subtle differences. Both kinds of replication middleware require six major infrastructure components:

- A *Multicast Service* for reliable, ordered dissemination of operations.
- A *Monitoring Service* for detection of faults in the system (e.g., crash of a service).
- A *Replication Manager*, mainly for maintenance of object/service groups and overall configuration of the replication logic.
- A *Replication Protocol* unit for providing the actual replication logic (e.g., primary-backup protocol).
- An *Invocation Service* providing the invocation logic (interception of client invocations, conveyance of the transaction context, etc.)
- An optional *Transaction Service* for supporting transactions on replicated entities.

Minor differences between the components are caused by (i) the different granularity of objects and services, (ii) different transaction models, and (iii) different technology standards (e.g., CORBA vs. Web services standards) used in both worlds.

As long as replication techniques are needed that allow for (but are not limited to) strict consistency, we believe the service-oriented community will benefit from our comparison since it clearly shows that the wheel need not be re-invented: We recommend to take a look at the well-established replication solutions for distributed object systems and apply the same concepts and even the same system architecture to service-oriented systems.

Future research challenges for replication middleware in service-oriented systems are (i) heterogeneous administration domains (e.g., better support by standardizing interfaces and architectures) and (ii) dynamic composition of services.

6. ACKNOWLEDGEMENTS

This work has been partially funded by the European Community under the FP6 IST project DeDiSys (Dependable Distributed Systems, contract number 004152, http://www.dedisys.org).

7. REFERENCES

[1] JGroups: A toolkit for reliable multicast communication. http://www.jgroups.org.

[2] Arjuna, Fujitsu, IONA, Oracle, and Sun Microsystems. Web services composite application framework ws-caf ver 1.0, 2003. http://developers.sun.com/techtopics/webservices/wscaf/primer.pdf.

[3] H. Attiya and J. Welch. Sequential consistency versus linearizability. *ACM Transactions on Computer Systems*, 12(2):91–122, 1994.

[4] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[5] O. Babaoglu, A. Bartoli, V. Maverick, S. Patarin, J. Vuckovic, and H. Wu. A framework for prototyping J2EE replication algorithms. In *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, volume 3291 of *Lecture Notes in Computer Science*, pages 1413–1426. Springer, Jan. 2004.

[6] A. Bessani, J. da Silva Fraga, L. Lung, and E. A. P. Alchieri. Active replication in CORBA: Standards, protocols, and implementation framework. In *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, volume 3291 of *Lecture Notes in Computer Science*, pages 1395–1412, Jan. 2004.

[7] S. Beyer, F. Muñoz-Escoi, and P. Galdámez. CORBA replication support for fault-tolerance in a partitionable distributed system. In *Proceedings of the 17th Int. Workshop on Database and Expert Systems Applications (DEXA)*, pages 406–412, Washington, DC, USA, 2006. IEEE Computer Society.

[8] K. Birman. The untrustworthy web services revolution. *IEEE Computer*, 39(2):98–100, 2006.

[9] N. Budhiraja, K. Marzullo, F. Schneider, and S. Toueg. The primary-backup approach. In S. Mullender, editor, *Distributed systems*, chapter 8, pages 199–216. ACM Press, Addison-Wesley, 2nd edition, 1993.

[10] G. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: a comprehensive study. *ACM Computing Surveys*, 33(4):427–469, 2001.

[11] F. Cristian. Understanding fault-tolerant distributed systems. *Communications of the ACM*, 34(2):56–78, 1991.

[12] P. Felber and P. Narasimhan. Experiences, strategies, and challenges in building fault-tolerant CORBA systems. *IEEE Transactions on Computers*, 53(5):497–511, 2004.

[13] Gorda Consortium. D1.1 - state of the art in database replication. Technical report, Universidade do Minho, Braga, Portugal, 2005.

[14] M. Hayden and K. Birman. Probabilistic broadcast. Technical report, Ithaca, NY, USA, 1996.

[15] L. Juszczyk, J. Lazowski, and S. Dustdar. Web service discovery, replication, and synchronization in ad-hoc networks. In *Proceedings of the 1st International Conference on Availability, Reliability and Security (ARES'06)*, pages 847–854. IEEE Computer Society, 2006.

[16] D. Liang, C.-L. Fang, C. Chen, and F. Lin. Fault tolerant web service. In *Proceedings of the 10th Asia-Pacific Software Engineering Conference*, pages 310–319. IEEE Computer Society, 2003.

[17] M. Merideth, A. Iyengar, T. Mikalsen, S. Tai, I. Rouvellou, and P. Narasimhan. Thema: Byzantine-fault-tolerant middleware for web-service applications. In *Proceedings of the 24th Symposium on Reliable Distributed Systems*, pages 131–142. IEEE Computer Society, 2005.

[18] L. E. Moser, P. M. Melliar-Smith, and P. Narasimhan. Consistent object replication in the eternal system. *Theory and Practice of Object Systems*, 4(2):81–92, 1998.

[19] Object Management Group (OMG). Common Object Request Broker Architecture: Core Specification, v3.0.3, 2004.

[20] J. Osrael, L. Froihofer, K. M. Goeschka, S. Beyer, P. Galdamez, and F. Munoz. A system architecture for enhanced availability of tightly coupled distributed systems. In *Proceedings of the 1st International Conference on Availability, Reliability and Security (ARES'06)*, pages 400–407. IEEE Computer Society, 2006.

[21] J. Osrael, L. Froihofer, G. Stoifl, L. Weigl, K. Zagar, I. Habjan, and K. Goeschka. Using replication to build highly available net applications. In *Proceedings of the 17th International Workshop on Database and Expert Systems Applications (DEXA)*, pages 385–389. IEEE Computer Society, 2006.

[22] H. Reiser, M. Danel, and F. Hauck. A flexible replication framework for scalable and reliable .net services. In *Proceedings of the IADIS International Conference on Applied Computing*, volume 1, pages 161–169, 2005.

[23] J. Salas, F. Perez-Sorrosal, M. Patiño-Martínez, and R. Jiménez-Peris. WS-Replication: a framework for highly available web services. In *Proceedings of the 15th International Conference on the World Wide Web*, pages 357–366. ACM Press, 2006.

[24] A. Schiper. Group communication: From practice to theory. In *SOFSEM 2006: Theory and Practice of Computer Science*, volume 3831 of *Lecture Notes in Computer Science*, pages 117–136. Springer, 2006.

[25] F. Schneider. Replication management using the state-machine approach. In S. Mullender, editor, *Distributed Systems*, chapter 2, pages 17–26. ACM Press, Addison-Wesley, 2nd edition, 1993.

[26] W3C. Web services description language wsdl 1.1, 2001. http://www.w3.org/TR/wsdl.html.

[27] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso. Understanding replication in databases and distributed systems. In *Proceedings of the 20th International Conference on Distributed Computing Systems*, pages 464–474. IEEE Computer Society, 2000.

[28] J. S. Y. Amir, C. Danilov. A low latency, loss tolerant architecture and protocol for wide area group communication. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 327–336. IEEE Computer Society, 2000.

[29] X. Ye and Y. Shen. A middleware for replicated Web services. In *Proceedings of the 3rd International Conference on Web Services*, pages 631–638. IEEE Computer Society, 2005.