Experiences from Building Service and Object Replication Middleware

Johannes Osrael, Lorenz Froihofer, Karl M. Goeschka

Institute of Information Systems
Vienna University of Technology
Argentinierstrasse 8/184-1, 1040 Vienna, Austria
johannes.osrael|lorenz.froihofer|karl.goeschka@tuwien.ac.at

Abstract

Replication is a primary means to achieve fault tolerance in distributed systems. While replication techniques are well known and have been widely applied in distributed object and database systems, they have not yet been extensively used in service oriented systems. However, if the success of service oriented computing shall continue in critical settings, replication middleware will play a crucial role in service oriented infrastructures. Thus, we contribute with a discussion of our experiences in building distributed object and service replication middleware and present the main lessons learned. Our conclusions are drawn from several replication middleware implementations built upon J2EE, .NET, CORBA, and Axis2.

1 Introduction

Service oriented computing is an emerging computing paradigm utilizing services to support the rapid development of distributed applications in heterogeneous environments. Complex services provide novel problems and pose new challenges for many disciplines (e.g. enterprise application integration, worflow management, etc.). Definitely, fault tolerance is one of the most important ones. Fault tolerance ensures that a service failure can be avoided when faults are present in the system [4]. Redundancy is a prerequisite for fault tolerance. *Replication* of both hardware and software resources is one important means to introduce redundancy and thus to enable fault tolerance.

Replication in service oriented systems is rather in its infancy — although replication has been researched for decades in classical systems such as databases, file systems, or distributed object systems. However, sound replication solutions are urgently required for service oriented architectures to close the dependability gap currently faced [16, 8].

In the DeDiSys project (Dependable Distributed Systems, http://www.dedisys.org) we are working both on advanced replication mechanisms for distributed object systems and basic replication mechanisms for Web service based systems. Web services are currently the dominant approach for realizing service oriented architectures. The interface of a Web service is described in the Web Service Description Language (WSDL [10]). The XML-based messaging protocol SOAP [10] is used for communication between Web services. Many other Web services standards/specifications exist, which are for instance focused on addressing, security, coordination, etc.

In the last years, we have designed and implemented replication middleware for both Web services and traditional distributed objects. In this paper, we summarize our experiences and present the main lessons learned. The remainder of this paper is structured as follows: Section 2 introduces our middleware prototypes, namely three variants (.NET, J2EE, CORBA) of a distributed object replication middleware for balancing data integrity with availability and a replication middleware for Web services built upon the Java-based Axis2 SOAP engine [2]. Section 3 presents the five main lessons learned. Related work is presented in section 4 before we conclude in section 5.

2 Our middleware solutions

The lessons learned presented in this paper are based on the following replication middleware implementations:

DeDiSys object replication middleware: DeDiSys is a replication middleware [23] for explicit runtime balancing of data integrity against availability. It provides novel replication protocols such as the Primary-per-Partition Protocol [5] and Adaptive Voting [20] that allow temporarily relaxing data integrity and replica consistency during degraded situations (node or link failures) in order to enhance availability.

The DeDiSys replication middleware is targeted to distributed object systems and has been implemented upon .NET, J2EE, and CORBA.

Axis2-based service replication middleware: We have also implemented a primary-backup replication middleware [24] for Web services built upon the Java-based Axis2 SOAP engine [2]. The middleware is implemented as Axis2 module. Axis modules allow to extend the functionality of Axis in a flexible way. For instance, WS-Addressing [10], WS-Security [27], WS-ReliableMessaging [27] have also been implemented on top of Axis using the module concept [2]. Performance evaluations [24] of the middleware show the relatively low overhead of Web services replication if the number of replicas is small.

3 Lessons learned

In this section we present the main lessons learned during design and implementation of the middleware solutions presented in section 2.

3.1 Architectural commonalities

Although one of the main goals of service oriented architectures is inter-enterprise integration, replication as it is required and used in critical settings such as air traffic control is an intra-enterprise concern[1]. That is, replication is not applied across organizational boundaries. Thus, replicas of a certain service can reside in a homogenous environment and replication middleware can be optimized for a certain kind of Web services technology, e.g. our replication middleware presented in [24] is targeted to Axis2 Web services. Therefore, today's replication frameworks for Web services can reuse many of the concepts of traditional (homogenous) replication frameworks used in distributed object systems or database systems as we have argued in previous work [21], where we compared service and object replication middleware (e.g. FT-CORBA [19], WS-Replication [25], FT-SOAP [17], [30]) on an architectural level. In our comparison we have shown that only subtle differences exist which are primarily technology-dependent. Thus, we derived a technology-independent architecture pattern for both service and object replication middleware. Our middleware implementations discussed in section 2 closely follow this pattern.

[1]This, however, does *not* imply a LAN setting, but rather refers to the homogeneous administration and control.

Lesson 1: Both object and service replication middleware require the following 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.

3.2 Key component: group communication toolkit

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 primary-backup replication, a backup has to be promoted to a new primary replica if the original primary crashes. However, before such steps are taken, agreement on the membership of nodes in the system has to be achieved. That is, system entities need to agree on which nodes are operational and which are not. As Birman points out, "in many ways, agreement on membership is thus at the center of the universe, at least insofar high assurance computing is concerned" [7]. A group membership service (GMS) keeps track of membership changes of *dynamic* groups, caused by voluntary (join or leave) changes or failures (crashed or unreachable nodes). A *view* contains the current members of a group. Group members are notified about group membership changes by the GMS. A *primary component* membership service ensures total order of views, while concurrent views may exist in *partitionable* membership services.

Reliable multicast primitives are needed as well—both in object and service replication middleware, e.g. for propagation of updates from the primary to the backup replicas in case of primary-backup replication. Since group membership changes have to be taken into account when a multicast is sent to a group, reliable multicast services are typically combined with a group membership service and referred as view-oriented group communication systems [9]. Group communication systems provide multicast primitives to (object, process, service) groups with configurable delivery and ordering guarantees.

Examples for state-of-the-art group communication

toolkits are Spread [1], JGroups [15], or the newly proposed SOAP-based WS-Multicast toolkit [25], 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, [10]) interface. However, this could also be realized for the other toolkits. We have used Spread in all our middleware frameworks. For the Axis2 replication middleware, SOAP messages are serialized into a byte stream before they are propagated by Spread.

Lesson 2: Of course, the use of group communication toolkits is not mandatory for replication middleware. However, we strongly recommend their use since they significantly reduce the implementation complexity. For instance, active replication requires ordering of operations, which can already be provided by a group communication toolkit. Group communication primitives hide most of the (implementation) complexity of primary-backup replication as well: 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.

3.3 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:

.NET: The .NET version of the DeDiSys replication middleware uses the .NET remoting framework for injecting interceptors at the client and server side of the invocation chain. The advantage of this approach is that the infrastructure is readily provided by the .NET framework itself. A disadvantage is that all the replicated objects must extend the MarshalByRefObject class, thereby limiting the designer's options for inheritance.

CORBA: The CORBA-based DeDiSys replication middleware [6] is based on JacORB [13] and uses CORBA portable interceptors [19] to trigger the replication logic, without the need for client modifications. CORBA invocations can be intercepted both on the client and the server side at different interception points. For instance, client-side interceptors are used in DeDiSys to re-direct invocations to the primary replica.

J2EE: Our J2EE DeDiSys replication framework builds upon the JBoss application server [14]. JBoss already includes an invocation service with the possibility to register interceptors either server-wide or separately for each application. We have defined custom interceptors for replication purposes both in the client and server invocation chain.

Axis2: The SOAP engine Axis2 [2] allows the definition of customizable message interceptors, so-called "handlers". The Axis flow is divided into phases, which are processed in sequential order. A handler is associated with each phase, i.e. first the handler of the TransportInPhase is called afterwards the handler of the DispatchPhase, etc. For replication purposes, we have defined the "replicationPhase" and an associated "InFlowReplicationHandler". Incoming SOAP messages are cut out of the Axis IN-flow by the "InFlowReplicationHandler", propagated to the other replicas and injected in their IN-flow.

Lesson 3: All of the state-of-the-art technologies we have used provide many options for interception of invocations and allow custom-tailored extensions. This eases the implementation of the invocation logic of a replication middleware and helps to achieve replication transparency.

3.4 Replica management and replication protocol

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 [17], DeDiSys [23], FT-CORBA [19]). The *Web service replication component* of the WS-Replication framework [25] 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.

The actual replication protocol (i.e. triggering of update propagation, recovery, reconciliation, 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 [25]). 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.

.NET: Both the .NET replication manager and the replication protocol have been implemented from scratch using the C# programming language, since distributed object replication in .NET environments is rather a novelty. The replication manager and protocols should be capable of handling IMessage as invocation context because .NET Remoting provides the invocation logic.

CORBA: The CORBA-based replication manager internally uses CORBA's Portable Object Adapter (POA) for the association of objects with object references. Replication management and protocol interact with each other but are encapsulated in separate components in order to ease extensibility. More details can be found in [6].

J2EE: For the J2EE DeDiSys replication service, we build upon the replication framework of ADAPT [29]. For replication purposes, ADAPT provides an abstraction from a specific application server by providing a so called ComponentMonitor with events, e.g. afterCreate(), afterFind(), call(). The ADAPT J2EE replication architecture consists of two layers: the ADAPT replication framework and the replication algorithm layer. That is, different replication protocols can be plugged into the framework and can run on top of it.

Axis2: The replication manager for our Axis2 replication middleware has been implemented from scratch. The only subtle difference to distributed object replication managers is that the number of entities (services vs. objects) it maintains is typically smaller due to the coarse-grained nature of services compared to rather small-grained objects. The replication manager processes membership messages[2] sent out by Spread and takes appropriate action if required. The replication protocol component of the Axis2 replication middleware is primarily responsible for update propagation from the primary to the backups, in combination with the group communication toolkit Spread.

[2]Four different membership messages are distinguished: join, leave, disconnected, and network.

Lesson 4: While CORBA and J2EE (in combination with ADAPT) provide some support for replication out of the box, .NET and Axis2 require building the replication middleware and protocol from scratch. Our primary recommendation is to separate replication management and replication protocol. That is, the replication management unit should provide the basic mechanisms (e.g. location service functionality) that can be used for a variety of replication protocols while the protocol implements specific policies. In this respect no differences between service oriented and distributed object middleware arise.

3.5 Combining replication and transactions

Transactions are a fault tolerance technique by themselves; specifically the atomicity and durability properties of traditional ACID transactions are related to fault tolerance [26]. 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.

.NET: With .NET 2.0, the System.Transactions library was introduced that provides a native .NET implementation of transactions. Transactions can be either lightweight (inside an application domain) or they might employ the Microsoft Distributed Transaction Coordinator (MSDTC). So far we have not implemented transactional support in our .NET replication middleware since transactions are not required for the control engineering products of our industrial partner who uses the middleware.

CORBA: The CORBA-based DeDiSys transaction manager adheres to the Java Transaction API (JTA) [28] but has been implemented from scratch since network failures are often not properly treated by off-the-shelf transaction managers.

J2EE: The JBoss application server does not include a transaction system with support for distributed transactions in its current releases (4.x). Therefore, we have used the separate JBoss transaction service (JBossTS, http://labs.jboss.com/portal/jbosstm) with support for distributed transactions that can be manually integrated into the application server. This transaction service was acquired by JBoss from Arjuna, released as open source product and

will be the standard transaction service in future (5.x) releases of the application server.

Axis2: The Web services coordination framework (WS–Coordination [3]) provides a foundation layer for consensus between Web services, where specific consensus protocols can be built upon, e.g. distributed transactions. Two particular specifications for Web service transactions build upon the WS-Coordination framework: WS-AtomicTransaction [3] for short running ACID transactions and WS-BusinessActivity [3] for long running transactions with weaker guarantees.

Apache Kandula [2] is an open-source implementation of these specifications and is based on Axis. Unfortunately, Kandula2, which is targeted to Axis2 is currently in a preliminary stage. Thus, we have not yet tested it in combination with our replication middleware.

Lesson 5: While distributed object replication middleware often supports transactions (e.g. DeDiSys middleware [23]), support for transactions in replication middleware for service oriented systems is rather in its infancy. Up to our knowledge, only WS-Replication [25] has been combined with transactional support. Although this yielded promising results, there is clearly a need for further research in this area, especially with different replication protocols and other transaction models.

4 Related work

In previous work [21] we have compared state-of-the-art object and service replication middleware on an architectural level. However, in contrast to this paper, the previous one does neither provide details on our own middleware implementations with different technologies nor does it contain lessons learned from our prototypes. Our Axis2-based Web service replication middleware [24] did not even exist at the time of writing of the previous paper.

Besides our own work, up to our knowledge, no comparison of object and service replication middleware exists in literature, neither conceptual work as our previous paper [21] nor experience reports like this paper. Similarities between these two worlds with respect to replication are mentioned in [25] but not described. Although object replication middleware is well-established, middleware 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 typically contain brief comparisons, which however lack in-depth coverage. Moreover, many 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 middleware is often not described at all or only rather implicitly.

5 Conclusion

In this paper, we contributed with a discussion of our experiences gained during the design and implementation of object and service replication middleware based on .NET, J2EE, CORBA, and Axis2. The main lessons learned are the following:

Lesson 1: Service and object replication middleware share many architectural commonalities and only subtle technology-dependent differences.

Lesson 2: Group communication toolkits significantly reduce implementation complexity for the software engineer.

Lesson 3: State-of-the-art technologies provide many options for interception of invocations and allow custom-tailored extensions of the standard call flow. This eases implementation of the invocation logic of a replication middleware and helps to achieve replication transparency.

Lesson 4: Replication management and replication protocol should be separated in order to ease extensibility of the replication middleware.

Lesson 5: Combining replication with transactions is a major challenge for service replication middleware.

Based on our experience, we believe future research should be targeted to lesson 5, i.e. especially the application of novel transaction models (such as long running transactions) in service oriented systems with replicated services requires attention.

Moreover, although state-of-the-art Web service replication middleware frameworks address many of today's requirements for replication in service oriented settings—which is still not trivial—we believe systems of the future such as ultra-large-scale systems [11] will require additional research if replication in a "true" service oriented manner—especially with respect to heterogeneity—is required. Additional standardization efforts—similar to other horizontal protocols such as WS-Coordination [3]—for replication protocols, group membership services, group communication protocols, etc. are likely to be necessary for such settings [22].

6 Acknowledgements

This work has been partially funded by the European Community under the FP6 IST project DeDiSys (Dependable Distributed Systems, contract number 4152, www.dedisys.org).

References

[1] Y. Amir, C. Danilov, and J. Stanton. A low latency, loss tolerant architecture and protocol for wide area group communication. In *Proc. Int. Conf. on Dependable Systems and Networks*, pages 327–336. IEEE CS, 2000.

[2] Apache. Axis2, http://ws.apache.org/axis2/.

[3] Arjuna, BEA, Hitachi, IBM, IONA, and Microsoft. Web services transactions specifications, 2005. http://www-128.ibm.com/developerworks/library/specification/ws-tx/.

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

[5] S. Beyer, M. Bañuls, P. Galdámez, J. Osrael, and F. Muñoz. Increasing availability in a replicated partitionable distributed object system. In *Proceedings of the 4th Int. Symp. on Parallel and Distributed Processing and Applications (ISPA'06)*, volume 4330 of *LNCS*, pages 682–695. Springer, 2006.

[6] S. Beyer, F. Munoz-Escoi, and P. Galdamez. Corba replication support for fault-tolerance in a partitionable distributed system. In *Workshop Proc. 17th Int. Conf. on Database and Expert Systems Applications*, pages 406–412. IEEE CS, 2006.

[7] K. Birman. *Reliable Distributed Systems*. Springer, 2005.

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

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

[10] World Wide Web Consortium. http://www.w3.org.

[11] P. Feiler, R. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, L. Northrop, D. Schmidt, K. Sullivan, and K. Wallnau. *Ultra-Large-Scale Systems*. Software Engineering Institute Carnegie Mellon, 2006.

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

[13] JacORB. http://www.jacorb.org.

[14] JBoss. JBoss application server, http://www.jboss.org/products/jbossas.

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

[16] J.-C. Laprie. Resilience for the scalability of dependability. In *Proc. 4th Int. Symp. on Network Computing and Applications*, pages 5–6. IEEE CS, 2005.

[17] D. Liang, C.-L. Fang, C. Chen, and F. Lin. Fault tolerant web service. In *Proc. 10th Asia-Pacific Software Engineering Conf.*, pages 310–319. IEEE CS, 2003.

[18] L. E. Moser, P. M. Melliar-Smith, and P. Narasimhan. Consistent object replication in the eternal system. *Theor. Pract. Object Syst.*, 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, M. Gladt, and K. M. Goeschka. Adaptive voting for balancing data integrity with availability. In *On the Move to Meaningful Internet Systems 2006: Confederated Int. Workshops Proceedings*, volume 4278 of *LNCS*, pages 1510–1519. Springer, 2006.

[21] J. Osrael, L. Froihofer, and K.M. Goeschka. What service replication middleware can learn from object replication middleware. In *Proc. 1st Workshop on Middleware for Service Oriented Computing in conjunction with the Middleware Conf. 2006*, pages 18–23. ACM Press, 2006.

[22] J. Osrael, L. Froihofer, and K.M. Goeschka. On the need for dependability research on service oriented systems. In *Proceedings of the 37th Int. Conference on Dependable Systems and Networks*. IEEE CS, 2007.

[23] J. Osrael, L. Froihofer, K.M. Goeschka, S. Beyer, P. Galdámez, and F. Muñoz. A system architecture for enhanced availability of tightly coupled distributed systems. In *Proc. 1st Int. Conf. on Availability, Reliability and Security*, pages 400–407. IEEE CS, 2006.

[24] J. Osrael, L. Froihofer, M. Weghofer, and K.M. Goeschka. Axis2-based replication middleware for Web services. In *Proceedings of the Int. Conference on Web Services*. IEEE CS, 2007.

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

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

[27] OASIS standards. http://www.oasis-open.org/specs/.

[28] Sun Microsystems. Java Transaction API,http://jcp.org/en/jsr/detail?id=907.

[29] H. Wu, B. Kemme, and V. Maverick. Eager Replication for Stateful J2EE Servers. In *Proc. OTM Federated Conf.*, volume 3291 of *LNCS*, pages 1376–1394. Springer, 2004.

[30] X. Ye and Y. Shen. A middleware for replicated web services. In *Proc. 3rd Int. Conf. on Web Services*, pages 631–638. IEEE CS, 2005.