Die approbierte Originalversion dieser Dissertation ist an der Hauptbibliothek der Technischen Universität Wien aufgestellt (http://www.ub.tuwien.ac.at).

The approved original version of this thesis is available at the main library of the Vienna University of Technology (http://www.ub.tuwien.ac.at/englweb/).

Dissertation

Replication Techniques for Balancing Data Integrity with Availability

Ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Doktors der technischen Wissenschaften unter der Leitung von

o.Univ.Prof. Dr. Mehdi Jazayeri und
Dr. Karl M. Göschka

184-1
Institut für Informationssysteme
Arbeitsbereich für Verteilte Systeme
Technische Universität Wien

und

o.Univ.Prof. Dr. Gertrude Kappel

188-3
Institut für Softwaretechnik und Interaktive Systeme
Arbeitsbereich für Business Informatics
Technische Universität Wien

eingereicht an der Technischen Universität Wien

von

Dipl.-Ing. Johannes Osrael
Matrikelnr. 9926120
Kurze Zeile 64
2212 Großengersdorf

Wien, September 2007

_____

This work has been partially funded by the European Community under the Framework Programme 6 IST project DeDiSys (Dependable Distributed Systems, contract number 004152, http://www.dedisys.org).

Kurzfassung

Replikation wird in verteilten Systemen zur Verbesserung der Verfügbarkeit und Performanz verwendet. Die Konsistenz der Replikate und die Einhaltung der Datenintegrität (Konsistenz bezüglich von Integritätsbedingungen) sind Korrektheitskriterien in datenzentrierten verteilten Systemen. Falls Konsistenz immer eingehalten werden muss, verschlechtert sich die Verfügbarkeit des Systems in Fehlersituationen (Verbindungsfehler, Rechnerausfälle). Es gibt jedoch Systeme (z.B. in der Flugsicherung) in denen die Konsistenz temporär in Fehlersitutationen abgeschwächt werden kann, um die Verfügbarkeit zu erhöhen. Das heißt, Verfügbarkeit und Konsistenz können gegeneinander balanciert werden. Dies erfordert jedoch Reparaturmaßnahmen, wenn Netzwerkpartitionen wieder zusammengefügt werden. D.h., mit Hilfe von Reconciliation-Protokollen muss sowohl die Konsistenz der Replikate als auch die Datenintegrität wiederhergestellt werden.

Der Hauptfokus dieser Dissertation liegt auf Replikationstechniken, welche die Steuerung des Zusammenspiels zwischen Verfügbarkeit und Konsistenz ermöglichen. Ein zweiter Fokus liegt auf Replikationstechniken für datenzentrierte service-orientierte Systeme. Die Dissertation besteht daher aus drei Hauptteilen:

Erstens wird ein Replikationsmodell für die Balancierung von Verfügbarkeit und Konsistenz vorgestellt, das sogenannte Availability/Consistency Balancing Replication Model (ACBRM).

Zweitens wird Adaptive Voting präsentiert, ein konkretes Replikationsprotokoll, welches dem abstrakten Replikationsmodell folgt. Sowohl eine Analyse der Verfügbarkeit als auch eine Prototyp-Implementierung zeigen die Sinnhaftigkeit des Ansatzes, inbesonders wenn (i) eine signifikante Anzahl an Integritätsbedingungen aufweichbar ist und (ii) die Zeit zur Wiederherstellung der Konsistenz kürzer ist als die Zeit, in der Fehler auftraten.

Drittens werden Replikationsmiddleware für verteilte Objektsysteme (z.B. die auf das ACBRM zugeschnittene DeDiSys Middleware) und Replikationsmiddleware für service-orientierte Systeme auf Architekturebene verglichen. Basierend auf dieser Analyse und Erfahrungen mit unseren Prototypen stammt die Schlussfolgerung, dass viele etablierte Middleware-Architekturen in service-orientierten Systemen wiederverwendet werden können.

Zukünftige Arbeiten sollten sich auf Techniken zur Erreichung von Sicherheit und Zuverlässigkeit in service-orientierten Systemen fixieren, die durch Heterogenität, hohe Skalierbarkeitsanforderungen, und Dynamik charakterisiert sind.

Abstract

Replication is used in distributed systems to achieve higher availability and/or performance. Correctness criteria for data-centric distributed systems are replica consistency and data integrity (also called constraint consistency). If consistency needs to be ensured all times, such systems soon become (partially) unavailable if node and link failures occur. However, there exist applications (e.g., in air traffic control) in which consistency can be temporarily relaxed during degraded situations in order to achieve higher availability. Thus, consistency can be balanced against availability. This in turn requires repair actions after reunification of network partitions. That is, reconciliation is necessary to re-establish replica consistency and data integrity when the system becomes healthy again.

The main focus of this thesis is on replication techniques for controlling this trade-off in distributed object systems; a secondary focus is on replication techniques for data-centric service oriented systems. Thus, the contribution of this thesis is three-fold:

First, we introduce an enhanced replication model for trading data integrity against availability — the Availability/Consistency Balancing Replication Model (ACBRM).

Second, we present Adaptive Voting — a concrete protocol that realizes the abstract model. Both an analytical availability analysis and a prototype implementation show the feasibility of the approach, especially if (i) a significant portion of data integrity constraints of the system is relaxable and (ii) reconciliation time is shorter than degradation time.

Third, distributed object replication middleware systems (e.g., the DeDiSys middleware which is targeted to the ACBRM) and service replication middleware systems are compared on an architectural level. From this analysis and experiences with our middleware prototypes we conclude that many well-established replication middleware architectures can be reapplied in service oriented systems.

Future work needs to focus on dependability and security techniques for service oriented systems of the future characterized by cross-organizational heterogeneity, massive scale, and dynamicity.

Acknowledgements

First of all, Dr. Karl Michael Goeschka deserves my deepest gratitude for opening the world of research to me.

Second, many thanks to Prof. Mehdi Jazayeri, Prof. Gerti Kappel, and Prof. Francesc Muñoz for their advice.

Third, I am thankful to my colleagues at the university and in the DeDiSys project for providing a great working atmosphere, especially Lorenz Froihofer, Dietmar Schreiner, and Wolfgang Forster.

Fourth, I would like to thank the Master's thesis students — in particular Norbert Chlaupek, Martin Weghofer, and Matthias Gladt — who contributed to prototype implementations, testing, and measurements.

Last but not least, I would like to thank my girlfriend and my family for their support throughout my career at the university.

Contents

Abbreviations

ACBRM	Availability/Consistency Balancing Replication Model
ACID	Atomicity, Consistency, Isolation, Durability
ACM	Association for Computing Machinery
a.k.a	also known as
AV	Adaptive Voting
Conf.	Conference
CORBA	Common Object Request Broker Architecture
DeDiSys	Dependable Distributed Systems
EJB	Enterprise JavaBeans
FIFO	first in, first out
FT-CORBA	Fault Tolerant CORBA
GEU	Greatest Expected Utility
GC	Group Communication
GMS	Group Membership Service
IEEE	Institute of Electrical and Electronics Engineers
Int.	International
IOGR	Interoperable Object Group Reference
LNCS	Lecture Notes in Computer Science
POA	Portable Object Adapter
Proc.	Proceedings
ROWA	Read-One/Write All
ROWAA	Read-One/Write All Available
Symp.	Symposium
trans.	transactions
typic.	typically
wrt.	with respect to
UDDI	Universal Description, Discovery & Integration
WSDL	Web Services Description Language

Chapter 1

Introduction and Overview on Replication Techniques

After introducing the motivation, contributions, and structure of this thesis, this chapter gives an insight into the groundwork for this thesis — namely replication techniques. Traditionally, replication has been first applied in file systems and database systems. Later on, distributed object systems followed and since recently replication is becoming more and more important for service oriented systems due to increased dependability demands. Thus, in this chapter traditional replication techniques are discussed in section 1.2 before the application of these techniques in service oriented systems is shown in section 1.3.

1.1 Introduction

This section introduces the motivation, highlights the main contributions, and presents the structure of this thesis.

1.1.1 Motivation

Many of today's distributed systems including diverse areas such as e-commerce, health care, military command and control, control engineering, air traffic control, and transportation need to be highly dependable. Dependability is the "ability of a system to avoid service failures that are more frequent or more severe than is acceptable" [ALRL04]. Dependability is an integrating concept and comprises the following attributes [ALRL04]:

- *Availability* is the "readiness for correct service".
- *Reliability* is the "continuity of correct service".

- *Safety* is the "absence of catastrophic consequences on the user(s) and the environment".
- *Integrity* is the "absence of improper system alterations".
- *Maintainability* is the "ability to undergo modifications and repairs".

Dependability can be achieved by the combined utilization of fault prevention, fault removal, fault forecasting, and fault tolerance techniques. Fault prevention aims at software and hardware development methodologies to prevent the occurrence or introduction of faults. Fault removal techniques (e.g., verification, diagnosis, correction) reduce the number and severity of faults both during system development and system use. Fault forecasting techniques (e.g., qualitative and quantitative evaluation) estimate the present number, future incidence, and the likely consequences of faults. Finally, fault tolerance ensures that a service failure can be avoided when faults are present in the system [ALRL04]. Redundancy is a prerequisite for fault tolerance. *Replication* is one important means to introduce redundancy and thus to enable fault tolerance — especially in data-centric distributed systems. This thesis addresses fault tolerance, in particular replication.

Data-centric distributed systems have the main focus on data and the processing of data. State of the art in software engineering of data-centric systems is to use an object-oriented approach for design and implementation of the functionality. Relational database management systems (or their object-relational successors) are most often used for persistent storage of the data. The object-oriented approach in this case uses the notion of an object encapsulating data. Examples for data-centric distributed systems are booking engines, online market places, or social networking platforms. Replication in a data-centric object-oriented distributed system can thus be either performed on the data or object level. This thesis focuses on the latter. One correctness criterion for data-centric applications is data integrity constraints, such as value constraints, relationship constraints (cardinality, XOR), uniqueness constraints and other predicates. A system is *constraint consistent* [Sme04, SG04] if all data integrity constraints are satisfied.

Traditional replication protocols (partially) block in *degraded situations* (node and link failures) in order to guarantee that neither replica nor constraint consistency are violated. However, some applications exist where consistency can be temporarily relaxed in order to achieve higher availability. For instance, in some safety-critical systems (e.g., [Kue07]) or in some control engineering applications (e.g., [Hab07]) availability is more important than consistency.

The main focus of this thesis is on enhanced replication protocols and middleware systems that are required to support the run-time configuration of the trade-off between availability and consistency.

Recently, service oriented architectures have been more and more adopted by industry in many areas of computing. If the success shall continue and the service oriented computing approach shall be applied in critical, vital systems dependability needs to be ensured in these systems as well. That is, among others, replication will play an important role to achieve dependability. Replication has been a research topic for more than three decades in traditional domains such as database systems, file systems, and later in distributed object systems. Nevertheless, applying the well-known principles of replication in service oriented systems is still not trivial, though urgently required to close the dependability gap [Lap05] we currently face in large-scale, heterogenous (service oriented) systems and to continue the success of the service oriented paradigm in critical settings. As pointed out by Ken Birman [Bir06], "only replication can ensure access to critical data in the event of a fault."

Thus, the second focus of this thesis is on replication techniques for data-centric service oriented systems.

1.1.2 Contributions

The contribution of this thesis is three-fold:

First, a replication model called Availability/Consistency Balancing Replication Model (ACBRM) [1, 2][1] for trading data integrity against availability is defined.

Second, Adaptive Voting [3, 4], a concrete protocol that realizes the abstract model is presented. Both an analytical availability analysis and a prototype implementation show the feasibility of the approach, especially if (i) a significant portion of integrity constraints of the system are relaxable and (ii) reconciliation time is shorter than degradation time.

Third, distributed object replication middleware systems(e.g., the DeDiSys middleware which is targeted to the ACBRM) and service replication middleware systems are compared on an architectural level [5, 6, 7, 8, 9, 10]. From this analysis and experiences with our middleware prototypes we conclude that many well-established replication middleware architectures can be reapplied in service oriented systems.

The PhD thesis of Lorenz Froihofer [Fro07] is complementary to this thesis and is focused on middleware support for adaptive dependability through explicit runtime integrity constraints. In particular, Froihofer's thesis provides insight on constraint validation approaches and constraint consistency management

[1]Publications co-authored by the author of this thesis are denoted with numbers and mainly summarized in this thesis. In the remainder of this thesis, these publications are used without being referenced individually.

concepts.

1.1.3 Structure of this Thesis

The structure of this thesis is as follows: After the introduction, the most common traditional replication techniques are introduced and analyzed regarding their suitability for service oriented systems in chapter 1. The first main contribution — the Availability/Consistency Balancing Replication Model — is presented in chapter 2. The second main contribution — the Adaptive Voting replication protocol — is presented in chapter 3. The third main contribution — a comparison of replication middleware for service oriented and distributed object systems — is presented in chapter 4. Finally, related work, conclusions, and future work are presented in chapter 5.

1.2 Traditional Replication Techniques

A plethora of replication protocols has been proposed in the past decades — some of them having only subtle differences. Thus, before introducing the most important replication techniques, different aspects and classification criteria[2] are discussed.

1.2.1 Classification Criteria

Type of the replicated entity: First, the type of replicated entity — e.g., object, data item, file, process, service — needs to be distinguished. The replication techniques for different types of replicas share many similarities but also subtle differences, as for example discussed by Wiesmann et al. [WPS+00] for database replication and classical distributed systems (i.e., processes, objects) replication.

Conceptually, many of the traditional replication techniques such as primary-backup [BMST93] or active replication [Sch93] are widely applicable for different types of replicas as will be shown later.

Full vs. partial replication: Replication can be categorized into full and partial replication: Full replication means that the whole state is available at all replicas while partial replication means that each replica contains only a subset of the state. For example, in a replicated distributed object system, full replication means that all objects are available on all replicated nodes, while

[2]Several classification criteria can be found in literature (e.g., [HHB96, WSP+00, SS05]). The selected criteria are the most suitable ones in the context of this thesis.

partial replication means that some replica nodes contain only a subset of the objects. However, an object replica is still a whole object and not only a part of an object.

Optimistic vs. pessimistic replication: Pessimistic replication techniques restrict operations (e.g., in case of degradation or mobility) in a way that update conflicts cannot occur. Optimistic replication techniques on the other hand aim at enhanced availability by allowing replicas to diverge and updates to conflict. The drawback of optimistic techniques is that conflicts need to be resolved eventually, e.g., after degradation or when mobile nodes re-join. The Availability/Consistency Balancing Replication Model and the Adaptive Voting replication protocol presented in chapters 2 and 3 of this thesis fall into this category. Saito and Shapiro [SS05] provide an excellent survey on further optimistic replication techniques.

Operation transfer vs. state transfer: Updates can be propagated from one replica to the others (e.g., from a primary to the backup replicas) using operation transfer or state transfer. Although these terms are widely used in the literature and sometimes informally described (e.g., [SS05]), to our knowledge, no precise definition of the terms exists.

We consider the content of the update message to precisely define the terms: If the message contains all necessary information to update the replica unconditionally, it is called state transfer. On the other hand, if some state information of the receiving node has to be read by the recipient of the message in order to perform the update, it is called operation transfer. For instance, assume the state is represented by a single variable A. An update message $A_{new} = A_{old} - 5$ does not contain all state information to perform the update and thus has operation transfer characteristics. An update message $A_{new} = 3$ has state transfer characteristics since the update can be performed without reading any additional information not contained in the message. According to our definition, an update message $A_{new} = 4 - 3 + 1 * 2$ has still state transfer characteristics, although the message contains an operation.

If operation transfer is used, all previous updates must have been successfully processed — updates must not be lost. On the other hand, if the full state (e.g., the whole state of an object) is transferred, lost updates do not matter — only the last update is important.

Whether operation or state transfer is used has influence on the performance of update propagation. For instance, processing an operation on only one node and forwarding only the result might be faster than processing an operation on all nodes. Another important aspect with respect to the choice of operation or state transfer is determinism: The content of a state transfer message is deter-

ministic while an operation transfer might contain non-deterministic elements such as random() functions.

Eager vs. lazy update propagation: Another consideration for update propagation is the timing aspect: An eager (synchronous, blocking) update propagation policy ensures that the return message to the client (that initiates the update) is returned after all replicas (that need to be updated) are updated. Lazy (asynchronous, non-blocking) variants update only a subset of replicas immediately and defer update propagation to the other replicas. Thus, the eager variant ensures strict consistency of all replicas while the lazy variant yields better response time for the client at the cost of relaxed consistency. A special form of lazy update propagation techniques are probabilistic variants such as epidemic replication [DGH+87].

After these general considerations now the most important replication techniques, namely primary-backup replication, active replication, coordinator-cohort/update everywhere, and quorum consensus techniques are discussed.

1.2.2 Primary-Backup Replication

In the original primary-backup approach [AD76, BMST93], only one of the replicas — the primary — processes clients' requests and forwards the results via state transfer to the other replicas — the backups. Thus, this technique is also called *passive* replication. The advantage of primary-backup replication is that at least one replica (the primary) exists which has all updates. Moreover, ordering of operations is easy to achieve since all operations are directed to the primary. However, the primary replica might become overloaded. While a crash of backup replica does not require specific actions by the replication protocol, a crash of the primary replica requires reconfiguration since a new primary needs to be promoted.

Updates can be propagated either in a synchronous or asynchronous fashion to the backup replicas. In the first variant, replicas are always kept consistent and thus read operations can be performed on local copies. In the latter variant, read operations on backup copies might return stale values.

The first primary-backup variant has been proposed in 1976 by Alsberg and Day [AD76]. Since then, primary-backup replication has been widely applied in distributed object systems (e.g., FT-CORBA [OMG04]), file systems (e.g., Harp [LGG+91]), and database systems (see [Gor05] for a survey).

1.2.3 Coordinator-Cohort/Update Everywhere Replication

The *coordinator-cohort* replication model [BJRA85] is similar to the primary-backup model in the sense that only one replica — the coordinator — processes a client's requests and propagates the updates to the other replicas — the cohorts. However, the coordinator can change for every invocation. Coordinator-cohort replication requires distributed concurrency control (e.g., distributed locking), deadlock detection, and deadlock removal mechanisms. While coordinator cohort replication has been introduced in the context of distributed objects, its counterpart in the database world is often referred as *update everywhere* replication (e.g., [KA00]).

1.2.4 Active Replication

In active replication [Sch93], all replicas receive and process the clients' requests. Thus, replicas need to behave in a deterministic way (state machine). That is, given the same initial state, replicas need to reach the same final state if the same operations are processed in the same order at all replicas. Sources for non-determinism are for example calls to non-deterministic functions (e.g., time() or random()) or the scheduling of concurrently executing conflicting transactions. Thus, non-deterministic sources need to be avoided (e.g., by using a deterministic scheduler and by removing all non-deterministic function calls) if this technique shall be applied. Semi-active replication [PCD91] does not require deterministic behavior of replicas. Deterministic operations are processed by all replicas but only one of the replicas — the leader — processes non-deterministic operations and propagates the result to the other replicas — the followers.

In contrast to primary-backup replication, active replication is a symmetric solution — the same code runs on all replicas. Failures are transparent to the client as long as one replica receives the client invocations. The difficulty in active replication is to ensure that all replicas process the operations in the same order. Ordering guarantees might be weakened if semantics of operations such as commutativity of operations are taken into account.

Active replication is widely applicable, including distributed objects (e.g., FT-CORBA [OMG04]), file systems (e.g., RNFS [MS88]) and even databases (e.g., Uni/cluster [Con]).

1.2.5 Quorum Consensus Replication

Majority Voting [Tho79] is the simplest quorum consensus scheme. Both read and write operations are performed on a majority of replicas. Thus, every read

operation will contain at least one up-to-date replica.

In Weighted Voting [Gif79], a generalization of Majority Voting, each replica is assigned some number of votes. Whenever a read or write operation shall be performed, at least RQ (read quorum) or WQ (write quorum) votes must be acquired. Let the total number of votes be V. The following conditions must be met:

$$RQ + WQ > V \tag{1.1}$$

$$WQ > \frac{V}{2} \tag{1.2}$$

$WQ, RQ, V \in \mathbb{N}$ and $WQ, RQ \leq V$ are assumed[3]. Condition (1.1) prevents read-write conflicts while condition (1.2) prevents write-write conflicts.

Quorum consensus techniques allow to balance the cost of read against write operations by adjusting the sizes of the read and write quorum appropriately. Furthermore, in static quorum schemes (as weighted voting), where the quorums are not reconfigured in response to failures, no intervention is necessary when network failures are repaired or nodes recover; i.e., failures are masked.

As coordinator-cohort replication, quorum schemes require distributed concurrency control, deadlock detection, and deadlock removal mechanisms.

While quorum consensus techniques have been initially proposed for files and databases, their application is also feasible for object based systems as it has been shown in the DeDiSys project (see chapter 3).

1.3 Replication in Service Oriented Systems

In this section, replication in service oriented systems is addressed on a conceptual level and a contribution is made by analyzing replication protocols regarding their suitability for service oriented systems.

Software and service engineers that need to build fault-tolerant service oriented systems can benefit from this analysis, especially in the design phase of the system life cycle.

1.3.1 Types of Services

A service is a self-describing computational element that performs some function [Pap03], e.g., a flight booking service. The capabilities of the service are defined by a service description language such as the Web Service Description

[3]In this thesis, $\mathbb{N}$ denotes all positive natural numbers, i.e., zero is not included.

Language (WSDL [W3C]). With respect to state, we distinguish between stateless and stateful services (Fig. 1.1). With respect to the functionality of the services, business services and infrastructure services need to be distinguished.

Stateful vs. Stateless Services

A stateless service does not maintain state; thus, the actions performed by a stateless service only depend on the content of the invocation message. In contrast, the actions performed by a stateful service depend on the content of the invocation message and the state maintained by the service. A stateful service either encapsulates state (e.g., in objects) or keeps it external in a data store such as a file or database. The latter type of stateful services can be modeled as a stateless "access" service plus a stateful resource[4].

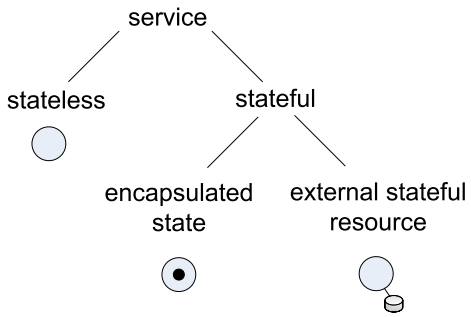

Figure 1.1: Types of services

Examples for stateless services are file compression/decompression services, temperature conversion services (e.g., Fahrenheit to Centigrade), file comparison services, etc. Many stateful services allow only read access to data: e.g., services for retrieving stock quotes, the current weather conditions, zip codes for a city name, etc. Most e-commerce services are stateful and require both read and write access to data, for example booking services (flights, hotels, cars, etc.). Besides accessing persistent data, many stateful services need to maintain conversational state [FFT+04], which can be either transient or persistent.

Business Services vs. Infrastructure Services

Service oriented systems typically contain both pure business services (as illustrated above) and infrastructure services. Business services can be divided into

[4]Thus, this category is also called "service that acts upon a stateful resource" [FFT+04].

atomic[5] (a.k.a. basic) services and composite services. While atomic services provide their functionality without interaction with other services, composite services are an aggregate of several services that — together — deliver a business function. A popular example for such a composite service is a booking engine of a travel agency, which combines several atomic services such as a flight booking service, a hotel booking service, and a car rental service. The service that coordinates these individual services to provide enhanced functionality is typically called an orchestration or composite service.

Examples for infrastructure services in a service oriented environment are directory services (e.g., UDDI registry [OAS]), messaging intermediaries (e.g., based on WS-Reliability [OAS]), and transaction coordinators. Business services often require such infrastructure services in order to provide their own functionality. For example, the above mentioned travel agency service may benefit from a transaction service.

1.3.2 Model of a Service Oriented System

Figure 1.2 presents a service oriented system with both stateful and stateless atomic and composite business services. Infrastructure services have been omitted in order to enhance the comprehensibility of the figure.

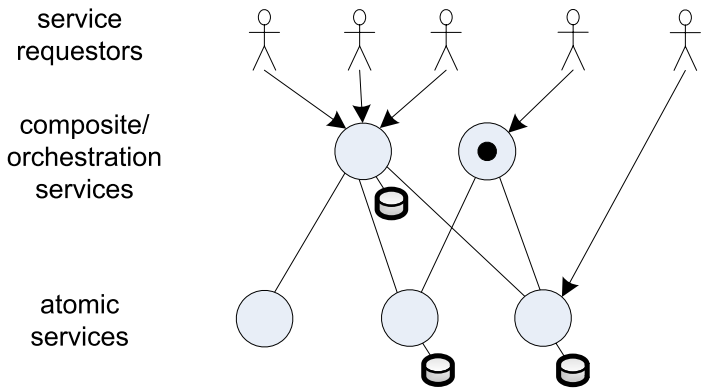

Figure 1.2: Service oriented system without replication

Figure 1.3 shows an extension of the system by introducing replication of both stateful and stateless services.

[5]Note, that an atomic service is a service which does not require other services. This has not to be mixed up with atomic transactions between several services.

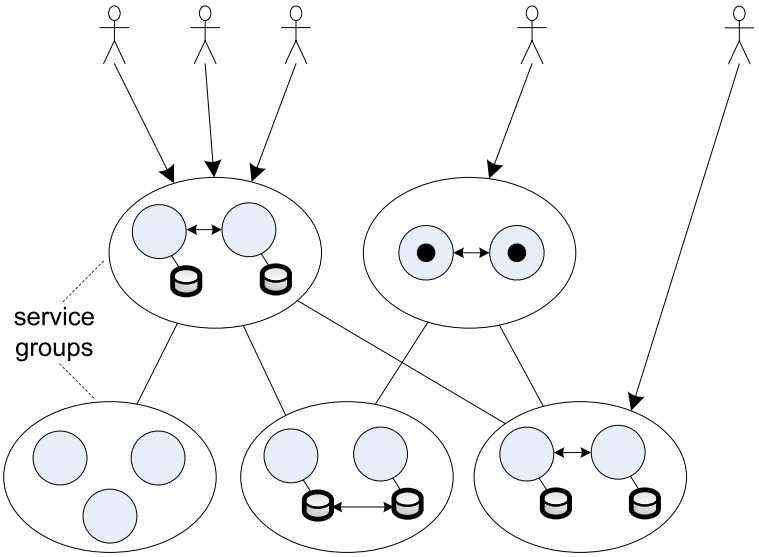

Figure 1.3: Service oriented system with replication on the service and data layer

Multiple instances of stateless services: "Replicating"[6] stateless services is comparatively easy (w.r.t. stateful services), since no state needs to be synchronized. Thus, stateless services are not further considered in this thesis. In principle, merely several instances of a service need to be deployed on different hosts in the distributed system. Nevertheless, middleware (see chapter 4) can be beneficial for keeping several instances of a stateless service, since it typically provides — besides facilities for the deployment — the abstraction of a logical service group (comprising several replicas), which reduces complexity for the application programmer.

Replication on the service and data level: Stateful services with an external stateful resource — typically a data store — can perform replication either on the service level or on the data level[7]. That is, replication can be performed on different layers in a service oriented system. In the first case, the state is synchronized by the "access" service while the underlying stateful resource (data store) takes care of state synchronization in the latter case. Both commercial database management systems (DBMS) such as Oracle [Ora], IBM DB2 Universal Database [IBM], or Microsoft SQL Server [Micb]

[6]The term replication is further avoided in the context of stateless services since — in the technical sense — it indicates the requirement of synchronizing state.

[7]In order to avoid a single point of failure on the service level, both the stateless "access service" and the underlying data store (stateful resource) should be replicated.

and open-source DBMS such as PostgreSQL [Pos] and MySQL [MyS] provide replication mechanisms [Gor05]. Distributed file management systems such as the Network File System (NFS) [SCR+03], Coda [SKK+90], or Microsoft's Distributed File System (DFS) [Mica] offer replication as well. Thus, if the stateful service builds upon a data store capable of replication, state synchronization of replicated services can be achieved on the data level via the data store.

1.3.3 Replication of Stateful Atomic Services

Now it is shown how the most important replication techniques (see section 1.2) can be applied for atomic services, on the service and data store level, respectively. Most of them can be configured for different consistency guarantees.

Primary-backup replication: In the *primary-backup* approach [BMST93], only the primary service receives a client's request. The backups are either updated on the service level (Fig. 1.4a) or directly by the underlying data store (Fig. 1.4b). S1 denotes instance 1 of service S. S1' is a replica of this instance. In the original primary backup variant, only the primary replica processes the requests and forwards the results to the backups. A slight variation is to forward the invocations instead of the results to the backups. Primary-backup replication can be performed in an eager or lazy fashion. An example of a primary-backup replication middleware for Web services is FT-SOAP [LFCL03] (see chapter 4).

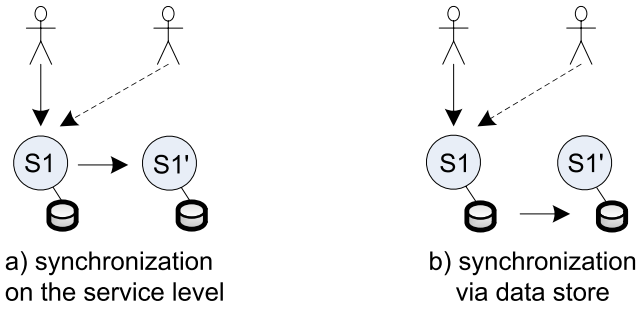

Figure 1.4: Primary-backup replication

Coordinator-cohort/update everywhere: As primary-backup replication, coordinator-cohort/update everywhere replication can be applied on the service or data level in a service oriented environment. To our knowledge, no

implementation of a coordinator-cohort replication protocol for Web services currently exists.

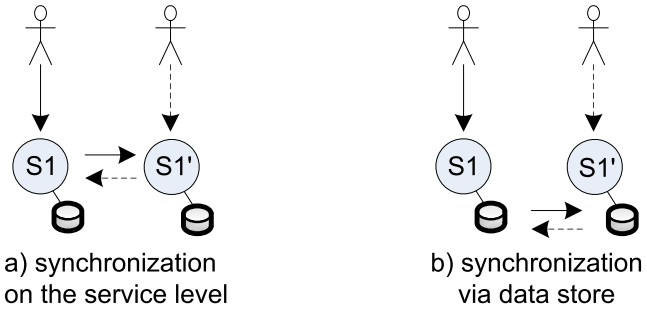

Figure 1.5: Coordinator-cohort/Update-everywhere replication

Active replication: In *active replication*, all invocations need to be issued in the same order to all replicas (total order) and services need to be deterministic. Active replication can also be performed on the data layer (Fig. 1.6b). In that case, the stateless part ("access service") of a stateful service becomes the client for the replicated data store.

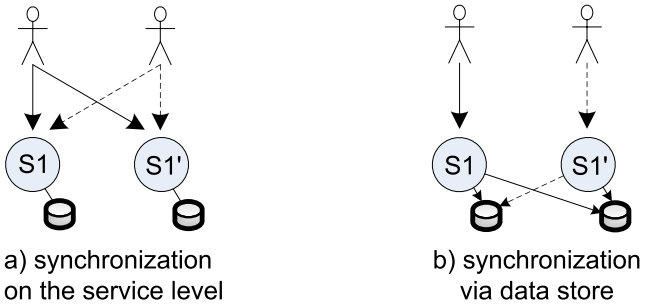

Figure 1.6: Active replication

WS-Replication [SPPJ06] is an example of an implementation of active replication for Web services (see chapter 4).

Quorum consensus replication: In principle, quorum consensus protocols can be used for replication on the data or service level of a service oriented system.

Epidemic replication: Epidemic replication protocols (e.g., as implemented in Bayou [TTP+95]) might be used in service oriented systems where a large number of replicas is needed, update conflicts rarely happen, and consistency

requirements are mild. To our knowledge, currently no implementation of an epidemic replication protocol for (Web) services exists.

1.3.4 Replication of Composite Services

Replication becomes more complex if the system is composed of several services that interact with each other as depicted in Fig. 1.2. In order to discuss various combinations of the most important (most widely used) replication methods (primary-backup, active, and coordinator-cohort/update-everywhere replication), without loss of generality, we consider a chain of two stateful services as depicted in Fig. 1.7a and thereby introduce the concept of layering with respect to replication. The client directs its request to the service on the upper layer which in turn needs to invoke the service on the lower layer in order to fulfill a task. It is also possible to perform replication on only one of the service layers or to use no replication at all.

No Replication at All: Figure 1.7 shows three different alternatives as to how a system without replication can be built. Figure 1.7a is the most obvious case with only one service instance at each layer. U1 is instance 1 of service U at the upper layer. U2 is instance 2 of this service. The service instances of the service L on the lower layer follow the same scheme. Although two instances of the same service type exist on the upper layer in Fig. 1.7b, this approach is *not* replication with respect to state since the state of the services is not synchronized, i.e., the services are of the same type but independent instances. Figure 1.7c shows how this concept is applied on both layers. This structure is only useful for stateless services or as building block in composite services with replication on only one out of several layers.

No Replication on the Lower Layer: Figure 1.8 shows combinations if replication is only applied on the upper layer but not on the lower layer. This situation can occur in service oriented environments if services with different availability/reliability guarantees (typically of different service providers) are combined: e.g., if a replicated flight booking service on the upper layer accesses a non-replicated weather information service on the lower layer.

U1' denotes the replica of instance 1 of service U on the upper layer. Coordinator-cohort replication is used on the upper layer in these examples. Figure 1.8a shows that no special treatment is necessary on the lower layer if replication is performed by the data store on the upper layer and only one service instance exists on the lower layer. In contrast, if invocations are replicated via service invocation on the upper layer as depicted in Fig. 1.8b, messages to

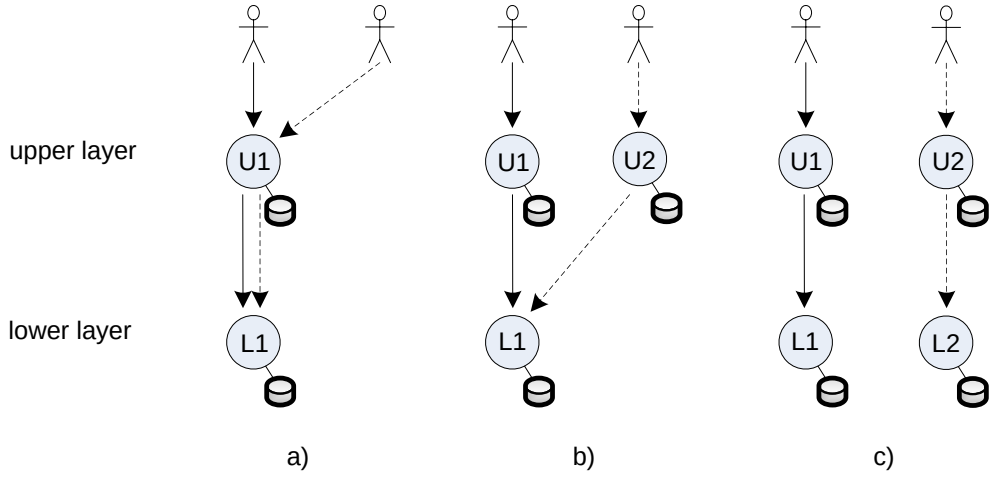

Figure 1.7: No replication at all

the lower layer are duplicated as well and hence need to be detected. This is called the *redundant nested invocation problem* [FLCL04].

Finally, Fig. 1.8c depicts what happens when each replicated service on the upper layer invokes a different, independent service on the lower layer: although these two service instances on the lower layer are not coordinated within their layer, consistency of the states can indirectly be established via coordination on the upper layer. That is, it behaves like active replication. This approach has been shown for the sake of completeness but is rather unlikely in practice.

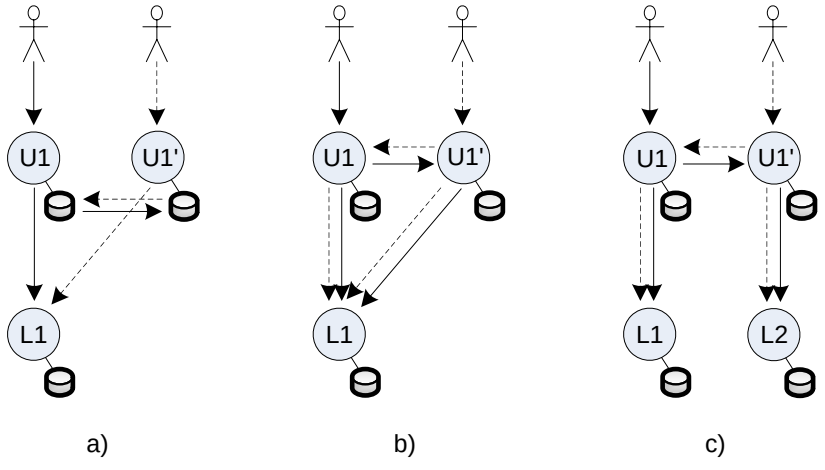

Figure 1.8: No replication on the lower layer

No Replication on the Upper Layer: Replication on the lower layer is not influenced, whether only one instance (Fig. 1.9a) or several independent, un-coordinated instances (Fig. 1.9b) of a service exist on the upper layer.

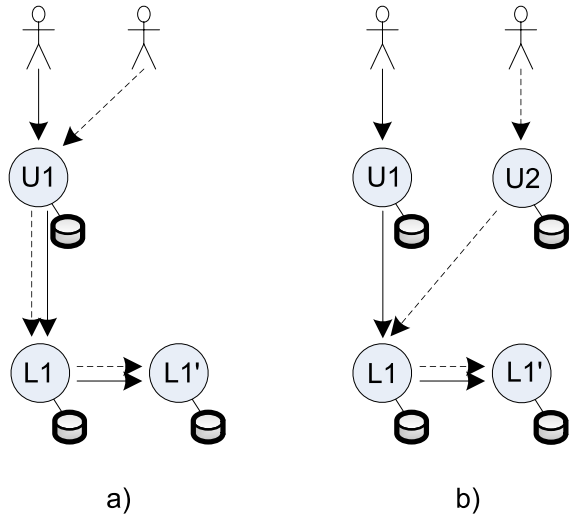

Figure 1.9: No replication on the upper layer

As in the previous case with no replication on the lower layer, such configurations are quite natural in service oriented systems, e.g., if services with different availability/reliability guarantees (typically offered by different service providers) are combined.

Replication on Both Layers: Figure 1.10 gives some examples how the fundamental replication techniques discussed in Sec. 1.3.3 can be combined: in Fig. 1.10a, the data stores on both layers perform primary-backup replication. In Fig. 1.10b, active replication is applied on the upper layer and primary-backup replication is applied by the data store on the lower layer. Replicated messages need to be detected if invocations are replicated on the upper layer. In Fig. 1.10c, update-everywhere replication is applied by the data store on the upper layer and active replication is used on the lower layer.

Other combinations and replication on additional layers follow the same principle. The most important subtlety that needs to be considered is redundant nested invocation detection if invocations are replicated.

Again, such combinations are quite natural in service oriented systems if services (of typically different service providers) are combined: e.g., as depicted in Fig. 1.10c, a Web service based application of an online bookshop (upper layer) might use update everywhere replication on the database level while the

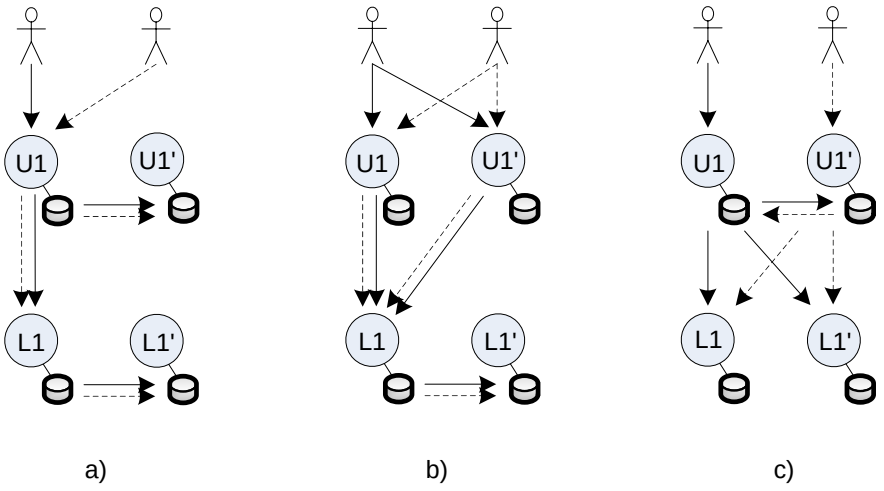

Figure 1.10: Replication on both layers

(via a Web service accessible) clearing system (lower layer) of the credit card company could use active replication.

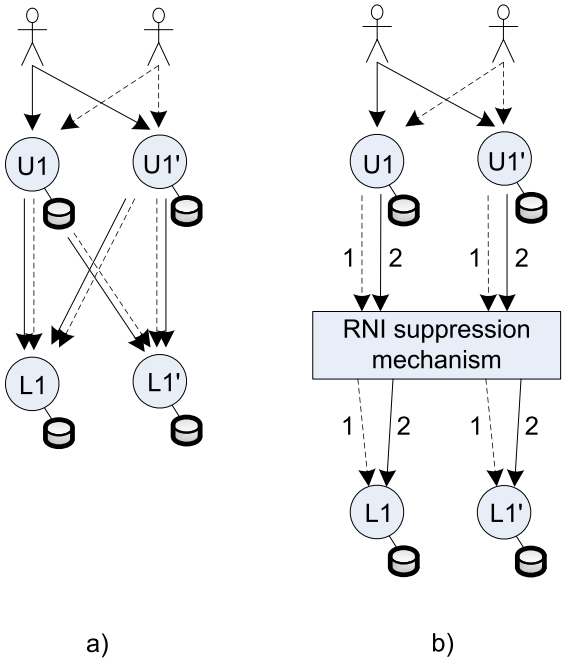

Figure 1.11: RNI suppression for single threaded services

Suppression of redundant nested invocations: A suppression mechanism for redundant nested invocations (RNI) needs to (i) automatically detect RNI, (ii) forward only one of the RNI and suppress the other RNI, and (iii) suppress redundant invocation replies and return the reply to all replicated services. For single-threaded services (i.e., one nested invocation is served at a time), the RNI problem can be solved by simply assigning sequential number identifiers to invocations. For example, suppose an actively replicated service U which in turn invokes an actively replicated service L.

Figure 1.11a depicts the situation without a suppression mechanism: the replicated instances of service L receive invocations twice. This leads to an incorrect state if the invocations are not idempotent, e.g., new state = old state + 4, assuming the state is represented by an integer value. Figure 1.11b depicts the solution: redundant invocations are detected by the RNI suppression mechanism by comparing the sequence numbers of the invocations. An invocation is suppressed if the sequence number has already been processed.

This solution also works for multi-threaded services (i.e., nested invocations are served in parallel) if a deterministic thread scheduler is used. If this is not the case, more sophisticated invocation identifiers (taking into account the thread contexts, etc.) need to be created in order to allow detection of identical invocations. Fang *et al.* [FLCL04] describe the redundant nested invocation problem in detail and provide a solution for multi-threaded actively replicated Web services.

Chapter 2

Availability/Consistency Balancing Replication Model

As discussed in chapter 1, enhanced replication techniques are required to support the run-time balancing of data integrity with availability. This chapter starts with a simple example to illustrate the balancing between these two requirements. Then, the notions of replica and constraint consistency are discussed in more detail and the system model is presented. Afterwards, the Availability/Consistency Balancing Replication Model (ACBRM) — which enhances a well-known abstract replication model to support the trading concept — is presented.

2.1 Trading Data Integrity against Availability

The most commonly used approach to improve availability in data-centric systems is to replicate data and services to several locations on the network, making at least one copy available while failures are present. Smeikal [Sme04] introduced the concept of temporarily relaxing data integrity to gain even more availability when link or node failures occur. Smeikal's solution for balancing data integrity with availability is targeted to the Distributed Telecommunication Management System [SG03] of the Vienna-based company Frequentis and uses a primary-backup replication approach. In contrast to the ACBRM replication model presented in this chapter and the concrete protocol Adaptive Voting presented in chapter 4, Smeikal's solution restricts write operations to one partition. However, the ACBRM (and Adaptive Voting) enhances availability even more by allowing non-critical updates in all partitions which requires resolving write-write conflicts during reconciliation time.

2.1.1 Ticket Booking System as Example

Figure 2.1 gives an example of a replicated system where this approach can be applied, a highly available ticket booking system. For the sake of simplicity but without loss of generality, it is assumed that the system comprises three server nodes that host replicas of two different classes of objects and two clients that perform operations on the replicas. In the example, one class of objects comprises all instances of a class Person and the other one all instances of a class Ticket (see Figure 2.2). The primary-backup approach [BMST93] is used as replication mechanism. Clients initiate write operations at the primary replica which propagates the update either as state or operation transfer to the other replicas (backups). Read operations can be performed at any replica since the replication mechanism ensures that all replicas are consistent during the absence of failures. Server node 1 hosts the primary copies in the example.

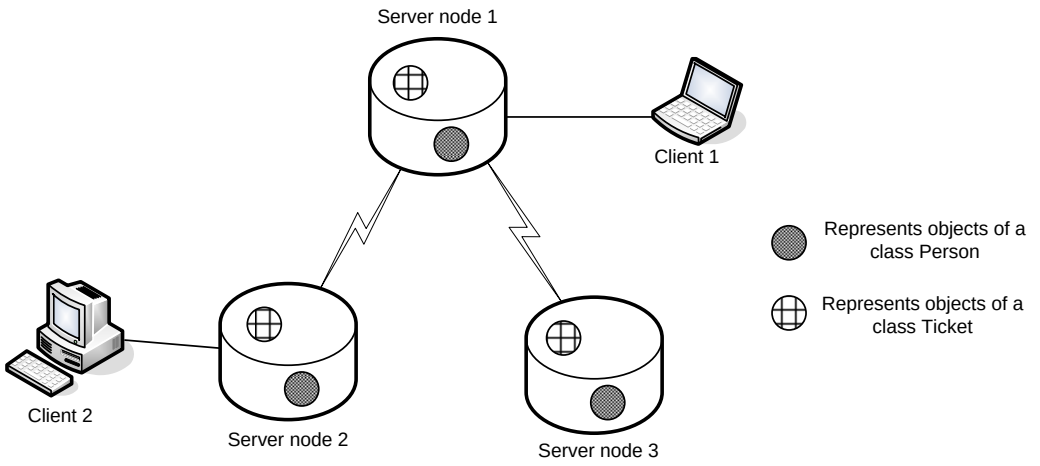

Figure 2.1: Ticket booking system

In order to demonstrate the trade-off between data integrity and availability we assume the cardinality constraint shown in Figure 2.2: A person might buy up to three tickets for the event and a ticket can only be owned by one person.

Healthy System

We assume a person called Bob has already bought one ticket. He wants to buy another ticket using client 2; hence, addTicket() is called on the object representing Bob. The cardinality constraint has to be checked before the new ticket can be sold to Bob. The operation succeeds since a person is allowed to own up to three tickets. However, the operation will be rejected if Bob wants to buy more than three tickets in total.

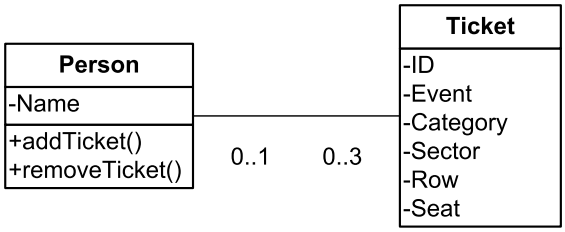

Figure 2.2: Class diagram booking system

Degraded System

A link failure occurred between server node 1 and server node 2. We assume, Bob, who has already bought two tickets before the network split happened, wants to buy a third one. Traditional systems, which do not allow inconsistencies, would block, because the primary copies are not accessible from client 2 due to the link failure. According to the replicas on server node 2, Bob has already bought two tickets. However, this information is potentially stale, since Bob could have bought a third ticket using client 1 in the meantime. The approach is the following: Bob is allowed to tentatively buy another ticket via client 2 and thus a consistency violation is risked. We call this situation a consistency threat [13], since the cardinality constraint might be violated if the operation is allowed. In order to allow revoking the purchase after repairing the degradation of the system, Bob is informed that the purchase is only tentative and the tickets will only be sent to him (by mail) if he did not already buy three tickets. Anyway, Bob ignores the warning and wants to buy a fourth ticket using client 1. According to the replicas on server node 1, Bob has only purchased two tickets yet. However, this information is potentially stale due to the network split and the same consistency threat arises as before. (In fact, the information is stale but the system cannot know this at this point in time.) We assume, the threat is accepted by the system and Bob is allowed to (tentatively) buy a fourth ticket. Hence, data integrity is temporarily relaxed to gain higher availability. However, data integrity will be re-established after the network split is repaired.

Reunification and Reconciliation

Reunification is the process of merging two or more network partitions on the network layer, i.e., communication between the partitions is re-established. As described before, our approach allows updates in different partitions to increase availability. Hence, reconciliation, the process of detecting and solving incon-

sistencies and conflicts caused by updates on replicas in different partitions, is necessary after reunification. In the example, the cardinality constraint is violated since a person is allowed to buy only three tickets but Bob bought four. The reconciliation strategy used in the example is to roll-back operations till the system becomes constraint consistent again: One of Bob's ticket bargains is rejected and only three tickets are sent to him. If new partitions and consistency threats arise during the reconciliation phase, a new trading process is started asynchronously. Meanwhile, reconciliation will continue as far as possible.

2.1.2 Replica and Constraint Consistency

Pessimistic replication schemes (partially) restrict operations during degraded situations in order to prevent update conflicts while optimistic replication techniques risk conflicts and re-solve them at repair time if they have occured [SS05]. Traditional replication protocols as primary-backup [BMST93], active replication [Sch93], and weighted voting [Gif79] fall into the first category, while our approach falls into the latter one. Pessimistic replication enables strict replica consistency. The notion of *replica consistency* denotes to what extent replicas differ from each other. However, strict replica consistency does not necessarily mean that *all* replicas need to be mutually consistent: For instance, in quorum consensus (voting) schemes it is sufficient that a consistent write quorum of replicas exists. Thus, replicas might diverge even in the case of strict consistency but no actions have to be taken when degradations are repaired.

Beside the two extremes — strict replica consistency and eventual replica consistency — numerous intermediate forms (e.g., [YV02]) of replica consistency exist which Saito and Shapiro [SS05] call *bounded divergence*. Our replication protocols fall into this category: Some operations — depending on the data integrity constraints they affect — are allowed in all partitions in order to enhance availability. Thus, replica consistency is relaxed and conflicts are risked in degraded situations. However, strict replica consistency is ensured in the healthy system and during degradation within a partition.

Whether or not replica consistency is relaxed by our replication protocols depends on the constraints that affect the replicated object. Some constraints of an application have to be satisfied at any point in time while others might be relaxed temporarily when failures occur. A system is called *constraint consistent* [Sme04, SG04] if all data integrity constraints are fulfilled. *Non-tradeable constraints* must never be violated while *tradeable constraints* can be temporarily relaxed during degraded periods.

Moreover, *preconditions, postconditions* and *invariant constraints* are dis-

tinguished. Pre- and postconditions have to be validated before/after a call to an operation and cannot be simply re-evaluated during repair time. Our replication protocols re-establish both replica and constraint consistency when failures are repaired, which requires re-evaluation of constraints. Hence, we focus on invariant constraints which are solely defined on the state of objects and can be validated at any time. [13]

We distinguish between *critical* and *non-critical* operations: Operations on objects affected by at least one non-tradeable (i.e., critical) constraint are called critical and operations on objects affected only by trade-able (i.e., non-critical) constraints are called non-critical. Only non-critical operations are allowed during degraded periods.

2.1.3 System Model

We focus on tightly-coupled, data-centric, object-oriented distributed systems with a small number of server nodes (typically 2-10) and an arbitrary number of client nodes. Server nodes host objects which are replicated to other server nodes in order to achieve fault tolerance. We consider both node and link failures (partitioning), i.e., the crash failure [Cri91] model is assumed for nodes and links may fail by losing but not duplicating or corrupting messages.

We assume a partially synchronous system, where clocks are not synchronized, but message time is bound. A group membership service is assumed in our system, which provides a single view of the nodes within a partition, i.e., it is used to detect node and link failures. Furthermore, we assume the presence of a group communication service which provides multicast to groups with configurable delivery and ordering guarantees.

We assume the correctness of the system is expressed in the form of application-specific data integrity constraints, which are defined upon objects that encapsulate application data (e.g., Entity Beans in Enterprise JavaBeans terminology). These objects do not contain business logic and typically correspond to a row in a table of a relational database. We assume that read and write operations on such objects can be distinguished, but we do not further differentiate invocations.

2.2 Key Concept of the Replication Model

Traditional replication protocols (partially) block in degraded situations, e.g., if the primary is not reachable in a primary-backup replication scheme or a quorum of replicas cannot be acquired in case of quorum consensus protocols. However, some systems do not require strict data integrity at all times, i.e.,

replica and constraint consistency can be temporarily relaxed during degraded situations.

Thus, the key idea is to enhance availability of traditional replication protocols by allowing non-critical operations in degraded situations in *all* partitions, even if replica conflicts may arise and data integrity constraints are possibly violated (threatened [13, 14]). Different *reconciliation policies* are required to re-establish replica and constraint consistency after nodes rejoin.

Our enhanced replication protocols distinguish three modes of operation: normal mode, degraded mode, and reconciliation mode. The current mode of the replication protocol depends on the system state, as it is locally perceived by each node (see Fig. 2.3).

Our replication protocols are in the *normal mode* when all nodes are reachable and all constraints are satisfied, i.e., no partitions are present and all repair activities (reconciliation) are finished.

The protocols switch into the *degraded mode* when not all nodes are reachable. Since node and link failures cannot be distinguished at the time the failure occurs [FLP85], node failures are treated as network partitions until repair time.

The protocols enter *reconciliation mode* when two or more partitions rejoin. The objective of reconciliation is to re-establish replica and constraint consistency of the system. System-wide consistency can only be re-established if all nodes are reachable. Thus, if partitions rejoin but the merged partition does not contain all nodes, either constraint consistency is re-established within the partition or constraint consistency is ignored and only replica consistency is re-established.

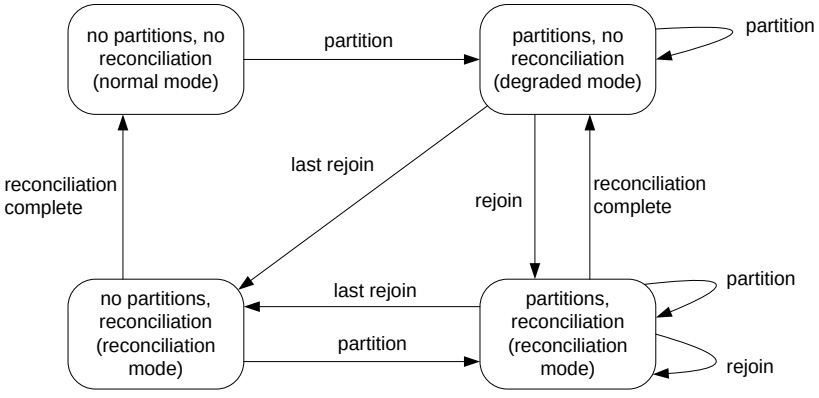

Figure 2.3: System states and protocol modes

2.3 The Model in Detail

The replication model is called *Availability/Consistency Balancing Replication Model (ACBRM)* and extends the functional model for replication protocols introduced by Wiesmann et al. [WPS+00] with respect to the trading of constraint consistency against availability.

In Wiesmann et al's model, replication protocols are described as a sequence of five generic phases. Literally from [WPS+00]:

- `Request (RE):` The client submits an operation to one (or more) replicas.
- `Server coordination (SC):` The replicas coordinate with each other to synchronize the execution of the operation (ordering of concurrent operations).
- `Execution (EX):` The operation is executed on the replicas.
- `Agreement coordination (AC):` The replicas agree on the result of the execution (e.g., to guarantee atomicity).
- `Response (END):` The outcome of the operation is transmitted back to the client.

Some of the replication techniques skip one or more phases, order them in another way, iterate over some, or merge some of the phases [WPS+00].

We extend the model of Wiesmann et al. by introducing new phases that enable the balancing of data integrity with availability. The new phases are only required for non-critical operations except the constraint validation phase which is also required for critical operations. Beside this phase, critical operations are treated as in Wiesmann et al's model.

2.3.1 Normal Mode

In normal mode, replication protocols for trading availability against consistency behave similar as traditional replication protocols, with the only extension that data integrity constraints are explicitly supported and validation of the constraints is triggered. Thus a new phase, called *constraint validation (CV)*, is required.

Constraint Validation

The constraint validation phase starts immediately after the *execution (EX)* phase of a write operation. Constraint validation is not performed in case of read operations.

Figure 2.4: Protocol phases in normal mode for write operations

The write operation is aborted — independent of the type of the constraint (tradeable or non-tradeable) — if one of the constraints is not met since our protocols guarantee data integrity in the healthy system. The CV phase is identical for all our concrete protocols.

An experimental evaluation of constraint validation approaches is presented in [15].

2.3.2 Degraded Mode

The key idea of our replication protocols is to allow non-critical operations in degraded situations in order to enhance availability, even if the consequence is that replica conflicts might arise and data integrity is threatened. Critical updates are treated as in normal mode.

Three new phases are introduced for the degraded mode of our protocols:

Configuration Adjustment (CA)

Replication protocols are configured with respect to various system parameters as the number of nodes, read to write ratio, load, etc. For example, in a quorum consensus scheme, the read and write quorums need to be configured. In a primary-backup approach, the roles (primary vs. backup) of the replicas have to be defined. Some replication protocols require reconfiguration in response to failures (e.g., of nodes or links) in order to provide fault tolerance on the system level, e.g., a new primary needs to be elected if the original primary crashes. In other protocols, as static quorum schemes, no intervention is necessary when failures occur, i.e., failures are masked.

Our protocols allow non-critical operations in all partitions during degraded situations. Thus, we adapt the protocols in degraded situations: For instance, in case of the Primary-per-Partition-Protocol [16], a temporary primary is elected in each partition for objects that are only affected by tradeable constraints. Configuration adjustment is partition-internal and must be based on partition-specific parameters as the number of replicas or the roles of the replicas residing in the partition.

Constraint Validation (CV)

For critical operations, constraint validation is performed as in normal mode. However, for non-critical operations, which are allowed in different partitions, constraint validation has limited significance: A tradeable constraint that is satisfied based on the objects in the current partition might be violated retrospectively if one of the involved objects is changed in another partition. Thus, it can be configured whether or not constraint consistency within the partition shall be enforced. In the latter case, tradeable constraints do not have to be validated in degraded mode but are marked (in the reconciliation preparation phase) for validation at reconciliation time.

Reconciliation Preparation (RP)

In order to allow maximum flexibility for reconciliation when nodes rejoin, the replication protocol needs to log information about non-critical updates during degraded mode. In principle, either operations or states can be logged. Furthermore, depending on the reconciliation strategy, it is required to log either all, some, or none of the operations and/or states. Logging everything (*full history*) offers all options during reconciliation but is the most resource-consuming approach. Keeping a *partial history* is a compromise between resource consumption and reconciliation flexibility and the third, *no history*, approach limits reconciliation to roll-forward or compensation actions that do not require a history of tentative operations/states.

Read operations do not require this phase: Read-write conflicts can be ignored during reconciliation since the application is aware that non-critical read operations performed during degraded situations return possibly stale objects.

Figure 2.5 depicts the sequence of the protocol phases in degraded mode for non-critical write operations. The configuration adjustment is the first phase and is triggered by the group membership service when the number of replicas in the current partition changes. However, not all changes require reconfiguration. For instance, CA can be skipped in case of the primary-backup scheme if the original primary is still in the partition. If constraint consistency shall be enforced within the partition, constraint validation is performed immediately after the write operation is executed. Preparation for reconciliation starts afterwards.

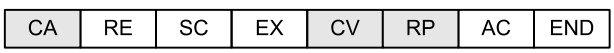

Figure 2.5: Protocol phases in degraded mode for non-critical write operations

2.3.3 Reconciliation Mode

Reconciliation mode starts after the new group view is established because of (partial) partition re-unification. The overall goal of reconciliation is to re-establish constraint consistency and replica consistency. Full consistency (i.e., system-wide) can only be re-established if all nodes are available (full constraint consistency re-establishment). If this is not the case, we consider two options: Either constraint consistency *and* replica consistency are re-established within the partition (partial constraint consistency re-establishment) or *only* replica consistency is re-established (partial replica consistency re-establishment). Our concrete replication protocols that follow the ACBRM allow to plug-in different reconciliation policies.

Full/Partial Replica Consistency Re-establishment (RCR)

If updates have occurred in only one partition, the updates of this partition are applied on a certain number of replicas (depending on the concrete protocol) in the merged partition.

Replica conflict detection: Replica conflicts — caused by updates in different partitions — can be detected based on syntactic and/or semantic information. Syntactic approaches use information about when, where, and by whom operations have been submitted. Examples for syntactic techniques are version vectors [JPR+83], time-stamps [Mad98], or precedence graphs [Dav84]. Semantic techniques [KRSD01] exploit properties such as commutativity or idempotency of operations. Our replication protocols use only syntactic information (version numbers) for detection of replica conflicts since syntactic approaches provide better scalability than semantic ones. Thus, conflicts are detected by comparing the version histories of the partitions. How the version histories are generated depends on the concrete protocol.

Replica conflict resolution: If conflicts have been detected, one of the conflicting updates is chosen and the other updates are discarded. Selection criteria are the number of updates, number of nodes in the partition, etc. Alternatively, a completely new or a default version can be installed to solve the conflict.

Re-establishment of replica and constraint consistency can be decoupled to reduce complexity. However, if both phases are combined, another selection criterion for replica conflicts is whether or not data integrity is satisfied by choosing one or the other version. If none of the replicas satisfies the constraints, one can be chosen according to the previously mentioned policies.

Full/Partial Constraint Consistency Re-establishment (CCR)

If at least one constraint is violated after re-establishment of replica consistency, the following policies can be distinguished to re-establish constraint consistency:

Re-schedule and replay: One option is to replay all tentative operations based on the last (replica and constraint) consistent state in a schedule that accepts as many operations as possible, under the conditions that both ordering and data integrity constraints are satisfied. This approach requires logging of all operations during degradation and suffers from scalability problems.

Stepwise rollback: Another option is to stepwise revert objects affected by the violated constraint to previous versions till the constraint is satisfied. In the worst case, all operations are undone. This approach requires logging of all tentative states in degraded mode (i.e., full history approach).

Compensation actions: In order to avoid time-consuming rollbacks or replays, application-specific compensation actions can be defined for some applications. For instance, a simple compensation action is to choose a default or completely new (agreed) version in case of a conflict. This approach is even possible if no history is maintained.

The chosen version that satisfies the constraints is applied at a certain number of objects (depending on the concrete protocol) in the merged partition. If constraint consistency is re-established system-wide, the version histories are cleaned and constraint re-evaluation flags are set to false.

Configuration Adjustment (CA)

The configuration of the replication protocols that follow the ACBRM is readjusted depending on the new situation after consistency is (partially) re-established. For instance, in the primary-backup scheme one of the two (temporary) primaries of the merged partition needs to be demoted to a secondary replica. In case of the adaptive voting protocol, the quorum sizes are adapted to the size of the partition.

Figure 2.6 depicts the sequence of the protocol phases in reconciliation mode. Replica consistency re-establishment is followed by constraint consistency re-establishment or might even be combined. Finally, the configuration of the protocol is adjusted (CA phase).

Figure 2.6: Protocol phases in reconciliation mode

2.3.4 Dependencies between Degraded and Reconciliation Mode

The number of reconciliation options depends on the reconciliation preparation phase as depicted in Fig. 2.7. Vice versa, reconciliation mode retrospectively influences the replication behavior in degraded mode. In case of the "reschedule/replay" and the "compensation actions" reconciliation approaches, the protocol behavior in degraded mode can be literally overwritten. For instance, it is even possible to retrospectively switch from a primary-backup based scheme to a voting algorithm. By applying a "stepwise rollback" reconciliation strategy, the effects of the degraded mode cannot be changed, though (partly) revoked.

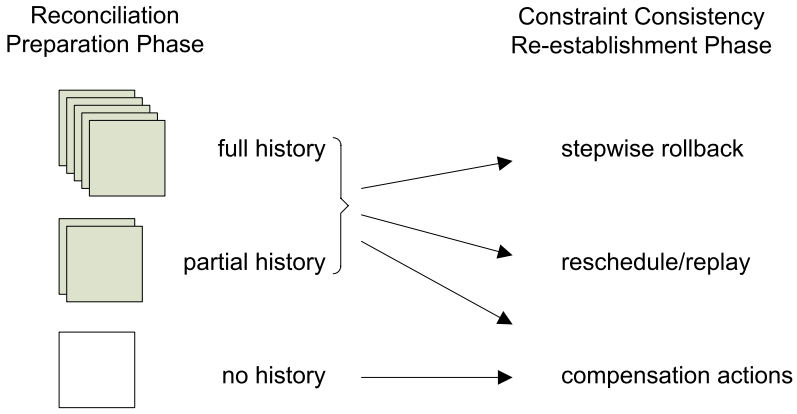

Figure 2.7: Dependencies between degraded and reconciliation mode

2.4 Proof of Concept

The proof of concept for this abstract model has been demonstrated by two concrete protocols adhering to the model. These protocols have been implemented in the DeDiSys middleware [5].

2.4.1 Concrete Protocols

So far, two concrete replication protocols that follow this model have been designed and implemented: The Primary-Per-Partition-Protocol [16], which is shortly described in this section, and Adaptive Voting [3]. The latter is described in detail in the next chapter since it is one of the main contributions of this thesis. In principle, other protocols such as coordinator-cohort [BJRA85] or active replication [Sch93] can be adapted in a similar way.

The Primary-Per-Partition-Protocol (P4) is defined as follows:

Normal mode: During normal mode, the P4 behaves like a traditional primary-backup protocol with the only extension that data integrity constraints are explicitly enforced. That is, clients send their requests to the primary replica which executes the request, validates the constraints, and propagates the updates to the backups if the constraints are met. Synchronous (eager) update propagation is used to ensure strict consistency in the healthy system, i.e., a response is sent to the client after the replicas are updated. In principle, however, the P4 could also use asynchronous update propagation.

Degraded mode: The P4 applies a primary-backup scheme per partition (thus the name primary-per-partition protocol) in case of network partitioning. That is, the P4 re-configures (configuration adjustment phase) once failures are detected and the primary is not in the current partition, i.e., a new temporary primary is elected for objects that are only affected by tradeable constraints. Thus, non-critical operations — that would not have been allowed in a traditional primary-backup scheme if the primary is not reachable — can proceed in all partitions. Tentative states and/or operations are logged in the reconciliation preparation phase.

Reconciliation mode: Conflicts between concurrent updates in different partitions which are merged can be easily detected by comparing the version histories of the two temporary primaries. The P4 allows to plug-in different reconciliation protocols (e.g., [AN06a, AN06b, ANBG07]).

2.4.2 DeDiSys

The ACBRM has been developed in the context of the European Union framework programme six research project DeDiSys (Dependable Distributed Systems, http://www.dedisys.org, contract number 004152). DeDiSys comprises eight partners from six countries and took place from October 2004 till November 2007. DeDiSys aims at optimizing dependability both in data-centric distributed object systems and resource-centric service oriented systems. The ACBRM and the concrete protocols that follow the abstract model are targeted to the first kind of systems.

The platform-independent system architecture of the DeDiSys replication middleware for data-centric distributed object systems, which is targeted to these adaptive replication protocols for balancing data integrity with availability is presented in chapter 4. The DeDiSys middleware has been implemented

on three different platforms: EJB [Kue07], CORBA [BMG06], and .NET [6]. The P4 has been implemented on all of these platforms. The Adaptive Voting (AV) protocol has been implemented for the .NET-based prototype [Chl07].

Two of the four industrial partners involved, namely Frequentis (Austria) and Cosylab (Slovenia), have been involved in implementation of the DeDiSys middleware and aim at exploiting DeDiSys results in their products.

Detailed test and validation reports of the DeDiSys middleware and three different industrial applications (ATS (Alarm Tracking System [Kue07]), ACS (Advanced Control System [Zag07]), and EPICS (Experimental Physics and Industrial Control System [Hab07]) Directory Service) that build upon DeDiSys can be found in [Kue07, Hab07, Zag07]. An experimental evaluation [Chl07] [4] of the Adaptive Voting protocol is discussed in the next chapter.

2.5 Summary

The Availability/Consistency Balancing Replication Model (ACBRM) — the first major contribution of this thesis — is an abstract replication model for balancing data integrity with availability. The ACBRM extends the abstract replication model proposed by Wiesmann et al. [WPS+00], which defines replication protocols as a sequence of five generic phases. Wiesmann et al.'s model is suitable to describe traditional replication protocols such as primary-backup [BMST93] or active replication [Sch93], but does not cover replication protocols for balancing data integrity with availability and thus needs to be extended for our purposes.

The ACBRM distinguishes three modes depending on the status of the system. Normal mode is reached if all nodes in the system are available and data integrity is established. Degraded mode is entered if some of the nodes are not reachable due to node crashes or link failures. During degraded mode, non-critical updates are allowed in all partitions which can lead to data integrity violations and replica inconsistency. Thus, a reconciliation mode is required to re-establish data integrity and replica consistency when network partitions rejoin.

The ACBRM defines the phases required in each of these modes for balancing data integrity with availability. The model is abstract in the sense that the phases are generic and need to be concretized for specific replication protocols. For instance, the ACBRM defines a configuration adjustment phase during degraded mode, which is different for primary-backup based protocols as the Primary-per-Partition-Protocol (P4) [16] and quorum-based protocols such as the Adaptive Voting (AV) protocol [3]. In the P4, temporary primaries need to be elected if the original one is not present in a partition. On the other hand, in the AV protocol, quorum adjustment denotes that the size of the read and

write quorums need to be changed during degradation in order to adapt to a changing partition size.

Adaptive Voting, one of the concrete protocols that follows the ACBRM, is the second major contribution of this thesis and described in the next chapter.

Chapter 3

Adaptive Voting

Chapter 2 presented an abstract replication model for balancing data integrity with availability. The model is abstract in the sense that it does not by itself define an implementable replication protocol. It does however define the principal phases required to support the run-time trading between availability and consistency. Concrete protocols that follow this model need to define the detailed behavior in each of the phases. In this chapter, one of the concrete protocols following the abstract model, namely Adaptive Voting, is presented.

Traditional Voting: Weighted Voting is described in section 1.2.5. For the sake of simplicity we further assume in this chapter that all replicas have equal votes (i.e., 1) and each node in the system hosts one replica. Thus, the total number of votes V becomes the total number of nodes in the system, denoted as N. We denote this simplification of weighted voting as *Traditional Voting*.

Key Concept of Adaptive Voting: Traditional voting blocks operations if the quorums cannot be built, hence, our key idea is to enhance availability of traditional voting by allowing non-critical operations even if no quorums exist, i.e., operations are allowed that may violate tradeable constraints but do not affect non-tradeable constraints. Furthermore, the new protocol called *Adaptive Voting* (AV) allows to re-adjust the quorums in degraded situations in order to support the tuning[1] of read against write operations. Since update conflicts and data integrity violations might be introduced, different policies are required to re-establish replica and constraint consistency after nodes rejoin. The replica consistency requirement for quorum consensus protocols is that a write quorum of replicas is consistent.

[1]The choice of the quorums depends on the read/write ratio and is not influenced by the data integrity constraints.

3.1 Protocol Description

In normal mode (healthy system), AV behaves as the traditional voting protocol with the enhancement that constraints are checked in case of write operations. That is, the quorum conditions

$$WQ_H + RQ_H > N \tag{3.1}$$

$$WQ_H > \frac{N}{2} \tag{3.2}$$

are obeyed. We denote the quorum sizes of the healthy system (i.e., all nodes are reachable) as WQ_H and RQ_H.

3.1.1 Degraded Mode

In traditional voting, read operations are allowed if a read quorum can be acquired and write operations if a write quorum can be acquired.

AV enhances availability by allowing non-critical operations even if no quorum exists. Critical operations are treated as in the normal mode to ensure that non-tradeable constraints are never violated.

Partition Size

The behavior of AV in degraded mode depends on the size of the partition. The number of nodes within a partition is denoted as P. Thus the number of nodes outside the partition is $N - P$. For the following considerations we assume that the quorums are not larger than necessary, i.e.,

$$WQ_H + RQ_H = N + 1 \tag{3.3}$$

which further implies

$$RQ_H \leq WQ_H \tag{3.4}$$

because from (3.3) follows $WQ_H + RQ_H - 1 = N$, which can be inserted into (3.2):

$$\begin{aligned} WQ_H &> N/2 = \frac{WQ_H + RQ_H - 1}{2} \\ 2WQ_H &> WQ_H + RQ_H - 1 \\ WQ_H &> RQ_H - 1 \\ WQ_H &\geq RQ_H \end{aligned}$$

Write and read quorum exist: If a write quorum exists in a particular partition, a read quorum exists as well. Outside this partition, no read or write quorum can exist. That is,

$$N > P \geq WQ_H \geq RQ_H \qquad \Leftrightarrow \qquad N - P < RQ_H \leq WQ_H \tag{3.5}$$

because

$$\begin{aligned} P &\geq WQ_H \\ P &> WQ_H - 1 \\ P &> N - RQ_H \\ -P &< RQ_H - N \\ N - P &< RQ_H \end{aligned}$$

where $WQ_H = N - RQ_H + 1$ follows from (3.3).

Both critical and non-critical operations are allowed within the partition. For critical operations, the behavior is as in the normal mode, i.e., those operations are blocked in other partitions. However, we allow non-critical updates in other partitions even if no write quorum exists. Thus, the quorum conditions are no longer satisfied system-wide and write-write or read-write conflicts might arise. However, we avoid partition-internal conflicts by using a quorum scheme within the partition. The (partition-internal) quorums can be adjusted according to the size of the partition in order to enhance performance. We denote the reduced quorums in the partition as WQ_P and RQ_P.

Since a read quorum RQ_H as defined in the healthy system exists, up-to-date copies of objects affected by non-tradeable constraints only can be retrieved. For objects affected by tradeable constraints, the read quorum might have been reduced in the partition. Thus, performance of the read operation can be improved by reading from RQ_P. However, since updates on objects affected by tradeable constraints are allowed in all partitions, the read operation might return an object that is *possibly stale*.

Write quorum does not exist but read quorum exists: If a read quorum but no write quorum exists in the partition, outside the partition no write quorum can exist but a read quorum may exist:

$$WQ_H > P \geq RQ_H \qquad \Leftrightarrow \qquad WQ_H > N - P \geq RQ_H \tag{3.6}$$

because

$$\begin{aligned} P &< WQ_H & \qquad P &\geq RQ_H \\ P+1 &\leq WQ_H & \qquad P &> RQ_H - 1 \\ P &\leq N - RQ_H & \qquad P &> N - WQ_H \\ N-P &\geq RQ_H & \qquad N-P &> WQ_H \end{aligned}$$

where $WQ_H = N - RQ_H + 1$ and $RQ_H = N - WQ_H + 1$ follow from (3.3).

Only non-critical operations are allowed in this situation. As mentioned before, the quorum sizes can be reduced for non-critical operations since valid quorums are guaranteed only within the partition anyway.
Thus, the following steps are performed:

1. If one of the constraints affected by the operation is non-tradeable, the update is not allowed. Otherwise the protocol acquires a write quorum WQ_P.

2. Afterwards, the operation can immediately be applied onto the write quorum, if constraint consistency within the partition is not enforced. Whether or not this is enforced is configurable in the AV protocol. If it is enforced, the operation will only be applied if no constraints are violated. A constraint check during degraded mode (for tradeable constraints) has limited significance anyway since the constraint might be influenced by operations in other partitions as well.

3. Thus, constraints affected by the operation need to be marked for re-evaluation at reconciliation time.

An object (and/or the operation) needs — depending on the reconciliation policy — to be saved in a version history before it is changed in degraded mode. This allows detection of update conflicts and provides several options for reconciliation.

Read operations are treated as in case (3.5).

Write and read quorum do not exist: If no read and write quorum exist in the partition, both a read and write quorum may exist outside the partition:

$$RQ_H > P \geq 1 \qquad \Leftrightarrow \qquad N - P \geq WQ_H \geq RQ_H \tag{3.7}$$

because

$$\begin{aligned} P &< RQ_H \\ P+1 &\leq RQ_H \\ P &\leq N - WQ_H \\ N-P &\geq WQ_H \end{aligned}$$

Write operations are treated as in case (3.6). Read operations can only be performed on a reduced read quorum RQ_P. Thus, all objects returned by a read operation are possibly stale. However, by applying the quorum conditions in the partition, it is guaranteed that subsequent read operations within a partition will return the same version.

Quorum Adjustment

AV allows updates in different partitions during degraded situations. However, within a partition, read-write and write-write conflicts shall be prevented and the tuning of read against write operations shall be supported. Thus, a quorum scheme adapted to the size of the partition is applied:

$$WQ_P + RQ_P > P \tag{3.8}$$

$$WQ_P > \frac{P}{2} \tag{3.9}$$

$$WQ_P, RQ_P, P \in \mathbb{N} \tag{3.10}$$

$$WQ_P, RQ_P \leq P \tag{3.11}$$

Different quorum adjustment policies can be distinguished:

Adjustment Policy 1: Maintaining read and write quorum: The most obvious strategy is to maintain the quorum sizes as in the healthy system as long as possible. If the partition size P falls below WQ_H (RQ_H), the write (read) quorum is set to P:

$$WQ_P = \min(WQ_H, P) = \begin{cases} WQ_H & : \quad P \geq WQ_H \\ P & : \quad P < WQ_H \end{cases} \tag{3.12}$$

$$RQ_P = \min(RQ_H, P) = \begin{cases} RQ_H & : \quad P \geq RQ_H \\ P & : \quad P < RQ_H \end{cases} \tag{3.13}$$

Figure 3.1 graphically shows this adjustment strategy. The horizontal axis denotes the size P of the partition, decreasing from the left side ($P = N$) to the right side (P=1). The vertical axis shows the sizes of the read and write quorums.

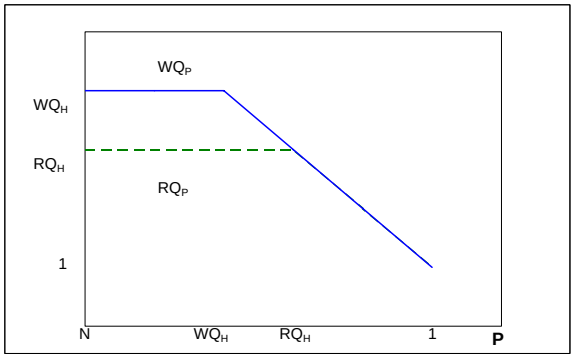

Figure 3.1: Maintaining write and read quorum

Adjustment Policy 2: Proportional adjustment: Adjustment policy 1 maintains the configuration of the healthy system as long as possible but with the cost that the quorums become larger than necessary within the partition. In order to maintain the read/write tuning of the healthy system, the read and write quorum in the partition can be adjusted proportional to the size of the partition. Since $RQ, WQ \in \mathbb{N}$, exact proportional adjustment is not always possible. Figure 3.2 graphically depicts the proportional adjustment strategy.

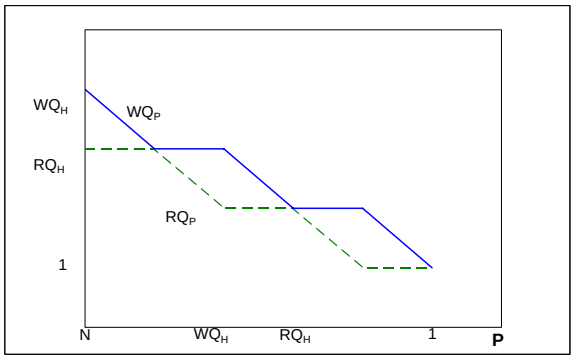

Figure 3.2: Proportional adjustment

Adjustment Policy 3: Arbitrary adjustment: In principle, all adjustments are allowed, as long as the above mentioned conditions (3.8), (3.9), (3.10), and (3.11) are met. Figure 3.3 graphically shows some arbitrary adjustment strategy.

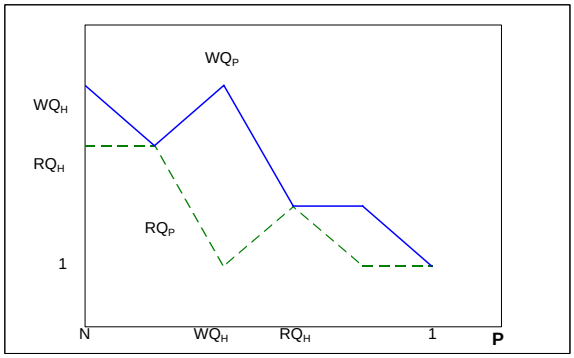

Figure 3.3: Arbitrary adjustment

3.1.2 Reconciliation Mode

In principle, different reconciliation protocols can be used for Adaptive Voting. General considerations regarding reconciliation in the Availability/Consistency Balancing Replication Model can be found in section 2.3.3. The default reconciliation strategy for AV is as follows:

The overall goal of reconciliation is to re-establish constraint consistency. However, full constraint consistency can only be re-established if all nodes are available. Thus, if this is not the case, AV only re-establishes replica consistency in the merged partition.

Non-critical operations are not allowed in reconciliation mode, therefore availability is reduced in this mode. Thus, reconciliation should be as fast as possible. In order to avoid combinatorial explosion, we use simple (application-defined) heuristics in the reconciliation phase, e.g., to select a particular version in case of a write-write conflict.
Reconciliation is performed in the following steps:

1. Re-adjustment of quorum sizes.
2. Re-establishment of replica consistency.
3. Re-establishment of constraint consistency if all nodes are available.

Re-adjustment of quorum sizes: The quorum size of the merged partition needs to be adjusted so that the appropriate quorum conditions are obeyed.

Re-establishment of replica consistency: AV allows non-critical updates in all partitions, even if no write quorum exists. Thus, write-write conflicts might arise. These conflicts can be detected by comparing the version lists of the partitions. If updates have occurred in only one partition, the version list of this partition is applied at a write quorum of the merged partition. In case of a conflict between the updates in the different partitions (we denote this as *replica conflict*), one of the replicas is chosen according to some pre-defined criterion (e.g., partition with more updates wins or larger partition wins etc.). The version list of the losing partition is discarded. The version list of the winning partition is adopted by a write quorum of the merged partition.

Re-establishment of constraint consistency: System-wide constraint consistency can only be re-established if all nodes are available. If this is the case, all constraints that are marked for re-evaluation are checked again. If a constraint is violated, the following policies are defined to re-establish data integrity:

1. Constraint conflict policy 1: Stepwise rollback: Objects affected by the constraint are stepwise reverted to previous versions till the constraint is satisfied.

2. Constraint conflict policy 2: Compensation actions: In order to avoid rollbacks, application-specific compensation actions can be defined. For instance, a simple compensation action is to choose a default version in case of a conflict.

The chosen version is applied at a write quorum of the merged partition. All tentative versions are discarded, i.e., the version lists are cleaned.

3.1.3 Example

Figure 3.4 gives an example of the behaviour of AV in normal mode, degraded mode, and reconciliation mode.

The system consists of 5 nodes. The quorums are identical for all objects: $WQ_H = 4$, $RQ_H = 2$. Two objects, namely object A and object B are replicated. For the sake of simplicity, the state of the object is represented by an integer value. Furthermore, to enhance readability of the figure, we assume the integer value is always equal to the version number, i.e., a write operation can increment A or B by 1. The following tradeable inter-object constraint is

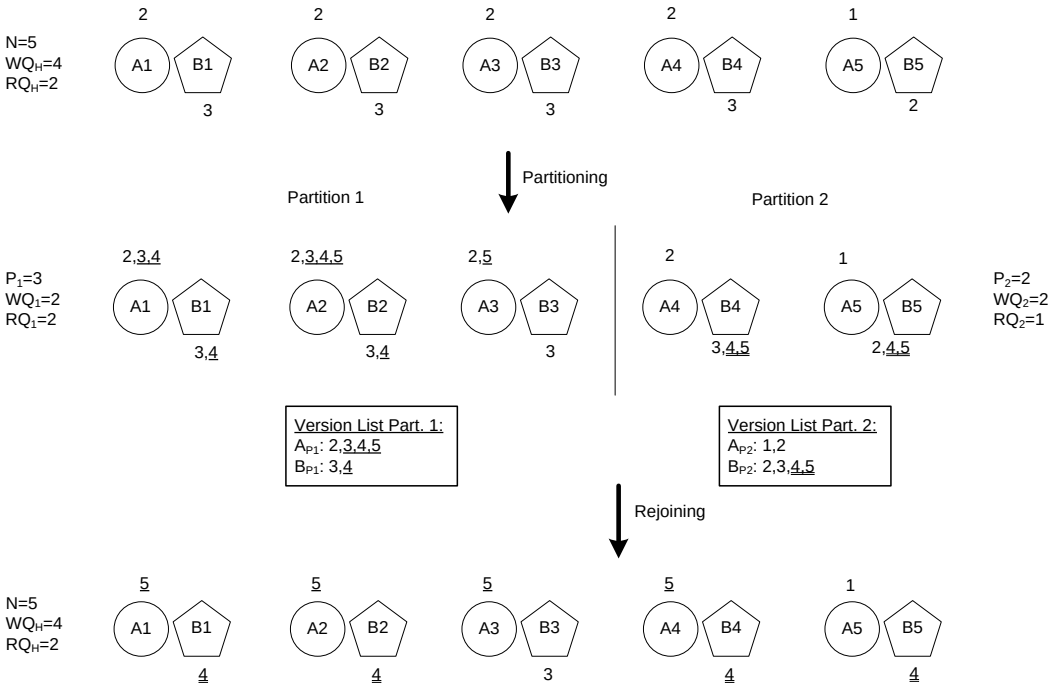

Figure 3.4: Example behaviour of AV

defined: $A + B < 10$. This constraint has to be fulfilled in the healthy system (normal mode). For instance, A is updated 3 times and B is updated twice in the healthy system.

The system degrades into 2 partitions; none of the partitions contains a write quorum. The group membership service detects the failure and AV changes to degraded mode. Since the constraint is tradeable, operations are allowed in all partitions during degraded mode. Within a partition, update conflicts are avoided by applying a quorum scheme adapted to the size of the partition. Thus, the quorums are changed in both partitions. We denote the reduced quorums in partition k as WQ_k and RQ_k. The size of partition k is denoted as P_k in Fig. 3.4. For partition 1, both WQ_1 and RQ_1 are set to 2. In partition 1, A is set to 3 first, afterwards to 4, then to 5. Furthermore, B is set to 4 in partition 1. In partition 2, B is set to 4 first and then to 5. The states are logged in order to allow reconciliation at repair time.

Reconciliation starts when two or more partitions rejoin. Again, this is detected by the group membership service. The quorums are changed to the initial quorums (i.e., WQ_H and RQ_H) since all nodes are available. Afterwards, the version lists — which can be built by reading from a read quorum — of the two partitions are compared. A has only been updated in partition 1, thus this

version is chosen. B has been updated once in partition 1 and twice in partition 2. Version 4 of B is chosen (which has the same value in both partitions) since version 5 would violate the constraint. Thus, B is set to 4 on a write quorum WQ_H of nodes. All tentative versions are discarded. AV returns to normal mode.

3.2 Availability Analysis

Jiménez-Peris et al. [JPAK03] compare various quorum schemes with the conventional read-one/write-all-available (ROWAA) approach in terms of availability, scalability, and performance. Regarding availability, they conclude that ROWAA is the best choice for a wide range of applications if no network partitions occur. However, if partitions are considered as in our target applications, ROWAA needs to adopt the primary partition approach and thus exhibits the same availability as majority voting [JPAK03]. Since majority voting is a special configuration of traditional voting, we compare availability of Adaptive Voting with availability of traditional voting (TV). For both protocols, availability of read and write operations needs to be distinguished. Total availability is expressed as

$$A = q_w A_w + q_r A_r \tag{3.14}$$

A_w is the write availability of TV and A_r the read availability for TV. Availability of AV follows the same scheme. q_w and q_r express the weight of write and read operations. E.g., $q_w = 0.4$ denotes that 40% of all operations are write operations. Thus, $q_w + q_r = 1$ must hold. Equal load on all nodes is assumed.

3.2.1 Traditional Voting

For TV, write/read availability is the sum of the availabilities in each partition. P_i is the size of partition i, N is the total number of nodes. One replica per node is assumed. Write operations can only be performed in at most one partition, while read operations might be performed in several partitions if no partition with a write quorum exists. Thus, the values of A_w and A_r in this case are as follows:

$$A_w = \begin{cases} \frac{\max_i P_i}{N} & : \quad \exists\, P_i \geq WQ_H \\ 0 & : \quad else \end{cases} \tag{3.15}$$

$$A_r = \sum_i \frac{P_i}{N} \qquad \forall P_i \geq RQ \tag{3.16}$$

3.2.2 Adaptive Voting

For AV, non-critical and critical write operations need to be distinguished. The latter type has the same availability (denoted as $\overline{A}_{w_{critical}}$) as write operations of TV. Availability of the first type — denoted as $\overline{A}_{w_{non-critical}}$ — is 100% in normal mode and in degraded mode while we assume[2] such operations are not allowed in reconciliation mode.

$$\overline{A}_{w_{critical}} = A_w \tag{3.17}$$

$$\overline{A}_{w_{non-critical}} = \begin{cases} 1 & : \ normal\ mode,\ degr.\ mode \\ 0 & : \ reconciliation\ mode \end{cases} \tag{3.18}$$

Thus, total availability of write operations of AV is expressed as

$$\overline{A}_w = q_c\, \overline{A}_{w_{critical}} + q_{nc}\, \overline{A}_{w_{non-critical}} \tag{3.19}$$

where $q_c + q_{nc} = 1$ must hold. q_{nc} is the percentage of non-critical write operations while q_c is the percentage of critical write operations.

Read operations follow a similar scheme: Read operations on objects affected by non-tradeable constraints have the same availability as read operations in TV. Read operations on objects affected by tradeable constraints have 100% availability in normal mode and degraded mode but are blocked during reconciliation mode.

3.2.3 Availability over Time

So far, we have analyzed the availability of the system in several system states, depending on the number and sizes of partitions. The analysis shows that AV provides higher (or at least equal if all constraints are non-tradeable) availability than TV in degraded mode. However, availability of AV declines during reconciliation. Thus, in order to decide when AV is beneficial, availability needs to be considered over time. We denote each t_i as a point in time where one or more nodes leave or rejoin. t_{ds} is the point in time where degradation starts and t_{re} where reconciliation ends. Availability is 100% in the healthy system for both protocols; thus, AV advances TV if

$$\frac{\sum_{i=ds}^{re-1} \overline{A}(t_i) \ \cdot \ (t_{i+1} - t_i)}{t_{re} - t_{ds}} \quad > \quad \frac{\sum_{i=ds}^{re-1} A(t_i) \ \cdot \ (t_{i+1} - t_i)}{t_{re} - t_{ds}} \tag{3.20}$$

[2]Only few reconciliation protocols serve business operations during reconciliation, e.g., the protocol described in [AN06b] maintains virtual partitions during reconciliation.

holds. That is, AV yields better availability over time if reconciliation time is short compared to degradation time and enough constraints are tradeable.

Figure 2 shows how availability might change over time. The figure is an example that has been chosen for presentation purposes but does not represent real measurements. In the beginning (healthy system), both TV and AV have full availability. Once partitioning happens, Adaptive Voting has only a slight availability decrease caused by critical operations that are not allowed during degradation. TV becomes completely unavailable if neither a write nor a read quorum can be built. On the other hand, availability of AV drops when nodes rejoin and reconciliation is performed.

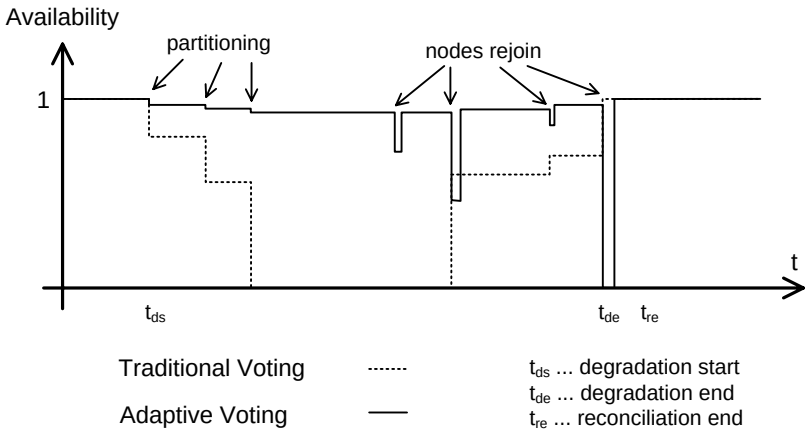

Figure 3.5: Availability over time: AV vs. TV

3.2.4 Influence of Reconciliation on Availability

Update conflicts and data integrity violations, which might be introduced in degraded mode since AV allows non-critical updates in all partitions, need to be resolved in reconciliation mode. Applying a rollback to a consistent checkpoint would revoke two types of operations. The first type would have been rejected in normal (healthy) mode anyway and thus their revocation would not retrospectively reduce availability. However, the second type of operations could have been applied successfully in normal mode either (i) as they are or (ii) as another operation according to the user's intention but based on the healthy context. If we do not want to reduce availability retrospectively at all, we have to assume that all update conflicts for the latter type of operations can be resolved by (i) replaying some operations or (ii) application-specific compensation actions. This is what we assume for our availability analysis. The first type of operations, however, can simply be revoked/undone. Putting

it the other way round: availability is only reduced retrospectively, if reconciliation revokes operations that could have been reasonably applied in the normal (healthy) mode. However, applying heuristics to cope with reconciliation complexity may further reduce availability retrospectively, which has to be investigated in future work.

3.3 Proof of Concept

AV has been implemented [Chl07] in the .NET version of the DeDiSys [5] replication middleware. DeDiSys is targeted to partitioned environments and allows to plug-in replication protocols for balancing data integrity against availability.

The following measurements [Chl07] [4] were conducted on a 100MBit full duplex switched network with up to ten machines with similar strengths (1-3GHz, 1-3GB RAM, Windows Server 2003). Spread [ADS00] has been used as group communication toolkit.

Three (independent) iterations of 1000 runs have been performed for every experiment and 1000 runs were performed before each iteration in order to reduce the effects of just-in-time compilation. All figures in this thesis show the average values over all iterations and runs.

The performance of Adaptive Voting has been evaluated using the following simple scenario: The state of an object a/b of class A/B is represented by an integer value x. Three constraints exist:

- C1: $a.x < constant1$
- C2: $b.x < constant1$
- C3: $a.x + b.x < constant2$

C1 and C2 are non-tradeable but C3 is tradeable.

3.3.1 Latency of Operations

Normal mode: Figure 3.6 compares the latency of (i) a write operation with constraint checking[3], (ii) a write operation without constraint checking and (iii) a read operation in normal mode for different node numbers. Adaptive Voting is configured with a read-one/write-all (ROWA) strategy in this figure, i.e., $WQ = number\ of\ nodes$, $RQ = 1$. AV applies a write operation locally first and then propagates the whole object state of around one kilobyte to a write quorum of replicas. Write operations without constraint checking are slightly

[3] cc = constraint checking

faster than write operations with constraint checking since the latter involves read operations for constraint validation as well. However, in case of ROWA, the difference is rather small since read operations can be performed locally. As expected from theory, latency of write operations increases with the number of nodes in a ROWA schema.

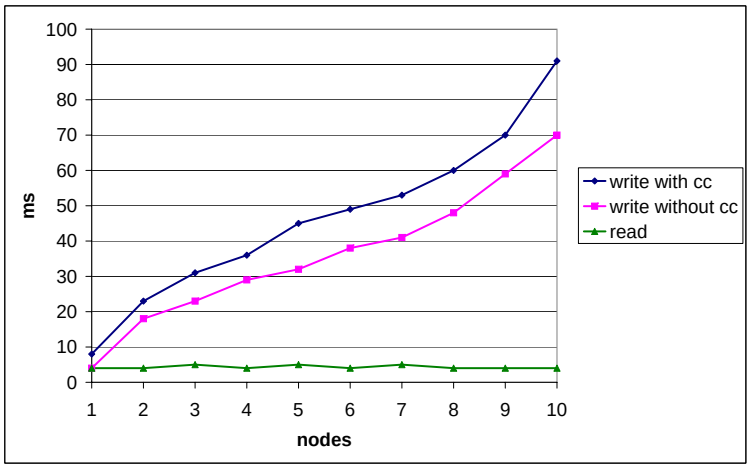

Figure 3.6: ROWA AV in normal mode [Chl07] [4]

Figure 3.7 shows the latency of these operations for Majority Adaptive Voting, i.e., $WQ = \left\lfloor \frac{N}{2} \right\rfloor + 1$ and $RQ = \left\lceil \frac{N}{2} \right\rceil$. Read and write operations without constraint checking have similar performance since the quorums are either identical (for an uneven number of nodes) or differ only by one (for an even number of nodes). Write operations with constraint checking are much slower since they involve remote read operations as well due to the inter-object constraint C3.

ROWA and Majority are the two extreme configurations of the Adaptive Voting protocol. While ROWA performs best for read operations, Majority is faster for write operations without constraint checking. Whether the one or the other strategy is better for write operations with constraint checking depends on whether information from remote nodes is required. In our example, which involves checking the inter-object constraint C3, ROWA would be the better strategy in terms of performance.

Besides these two extreme configurations, Adaptive Voting can be configured for other quorum sizes as well. Figure 3.8 compares the latency of operations for the possible quorum sizes if the number of nodes is ten:

In our example with an inter-object constraint, write operations with constraint checking perform best for the ROWA strategy since read operations required for constraint checking can be performed locally. The performance of write operations (without constraint checking) becomes slightly better with

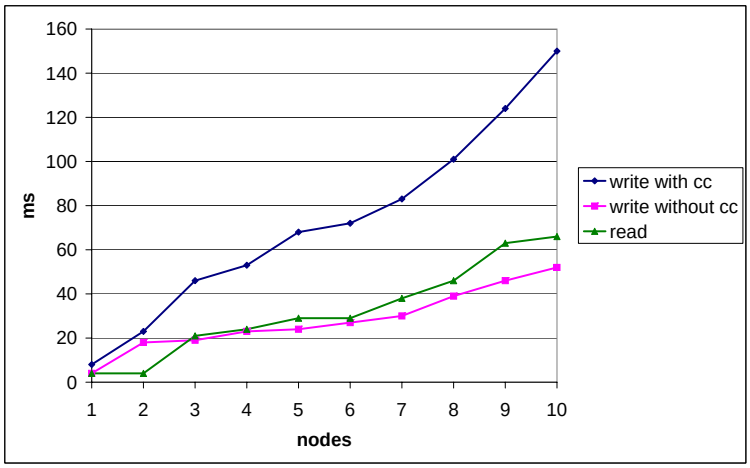

Figure 3.7: Majority AV in normal mode [Chl07] [4]

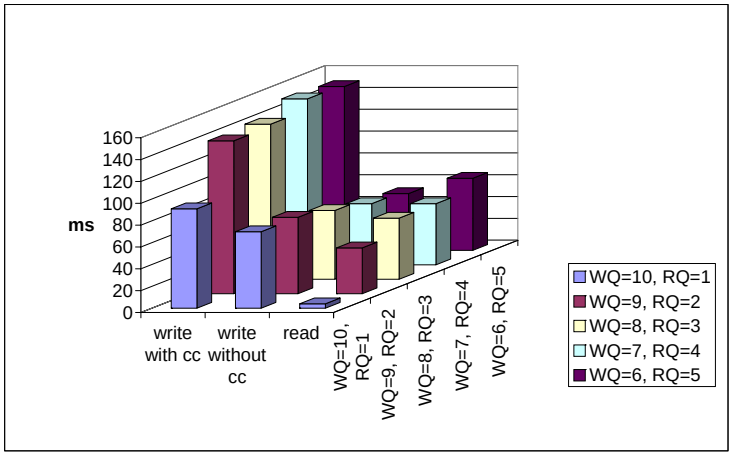

Figure 3.8: Quorum strategies for 10 nodes [Chl07] [4]

decreasing WQ. The trend for read operations is similar with decreasing RQ. That is, the figure shows that the performance of read and write operations (without constraint checking) can be balanced against each other.

Degraded mode: Read operations and critical write operations have similar performance in degraded mode and normal mode for a comparable number of nodes. Non-critical write operations are slower (for a comparable number of nodes) in degraded mode since logging of the operations and/or states is required. In our implementation the difference compared to normal mode is rather small since the logs are stored in memory and not in a database.

3.3.2 Performance of Reconciliation

The worst and best case for a rollback reconciliation strategy for the previously used simple scenario have been measured. Our system splits into two partitions containing three nodes each. The results depicted in Fig. 3.9 are from a configuration where a ROWA scheme is applied in both partitions and constraints are enforced within the partitions. Object a is updated in partition 1, object b is updated in partition 2. The best case for reconciliation is if the two partitions can simply be merged without violating the inter-object constraint. However, if the inter-object constraint cannot be fulfilled by simply merging the partitions, one partition is stepwise rolled back till the constraint is fulfilled — in the worst case to the initial state before degradation occurred.

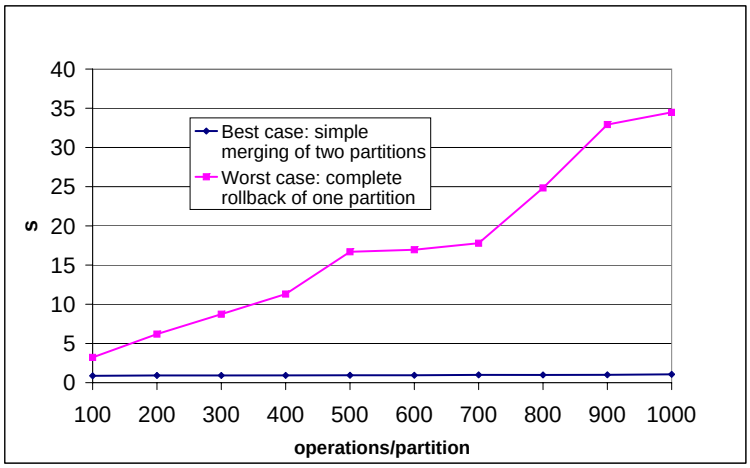

Figure 3.9: Example for reconciliation time [Chl07] [4]

Performance of reconciliation is highly application-specific, i.e., it depends on the constraints, failure pattern, load during degradation, and reconciliation policies. For instance, the more constraints need to be (re-)evaluated during reconciliation, the longer reconciliation will last. Reconciliation time often also drops if multiple partitioning has occurred or more operations have been accepted during degradation. Of course, the reconciliation policy per se (e.g., replay vs. rollback approach vs. compensation actions) has a significant influence on the performance of reconciliation as well.

3.4 Summary

Adaptive Voting — the second major contribution of this thesis — is a concrete replication protocol for balancing data integrity with availability and

follows the generic Availability/Consistency Balancing Replication Model presented in the previous chapter. Adaptive Voting is a quorum-based protocol that enhances availability by temporarily sacrificing data integrity during network partitioning. While traditional quorum based protocols such as weighted voting [Gif79] preserve consistency all time by blocking if a write quorum cannot be built, the Adaptive Voting protocol relaxes the quorum conditions for non-critical operations. That is, the quorums for non-critical operations are adjusted in each partition. This leads to an immediate increase of availability during degraded mode but might cause data integrity violations. Thus, reconciliation activities are required to re-establish data integrity and replica consistency when network partitions rejoin.

Both an availability analysis and a prototype implementation show the feasibility of the Adaptive Voting protocol, especially if some data integrity constraints are relaxable (and thus some non-critical operations can be allowed during degradation) and reconciliation time is shorter than degradation time.

Chapter 4

Replication Middleware Architectures

According to ObjectWeb [Obj07], "in a distributed system, middleware is defined as the software layer that lies between the operating system and the applications on each site of the system". The motivation for middleware is to re-use infrastructure code and avoid re-invention of the wheel for each application. Replication is one of the cross-cutting concerns that can be provided or supported by middleware for highly dependable systems.

In this chapter, the DeDiSys [5] replication middleware — which is targeted to the Availability/Consistency Balancing Replication model — and FT-CORBA [OMG04] are compared on an architectural level with replication middleware for service oriented systems. Finally, lessons learned from prototype implementations of both distributed object and service replication middleware are presented.

4.1 Replication Middleware for Distributed Objects

Research on fault tolerant distributed object systems has mainly focused on CORBA (Common Object Request Broker Architecture) [OMG04]. Thus, we have chosen FT-CORBA (Fault Tolerant CORBA) for comparison since it is a well-known standard. FT-CORBA supports both active and passive replication while DeDiSys supports the latter replication model. We do not consider fault tolerant CORBA-based systems (e.g., Eternal [NMMS97], AQuA [RBC+03], Maestro [VB98], FRIENDS [FP98], IRL [MMVB00]), which have been designed previously to the establishment of the FT-CORBA standard: As it is argued in [FN04], the FT-CORBA standard has been directly or indirectly

influenced by many of these systems.

Furthermore, we have chosen the DeDiSys distributed object middleware since it has been implemented on the three major middleware platforms J2EE [Kue07], Microsoft .NET [6], and CORBA [BMG06].

4.1.1 DeDiSys Middleware

Figure 4.1 shows the replication architecture of the platform-independent DeDiSys middleware [5]. 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.

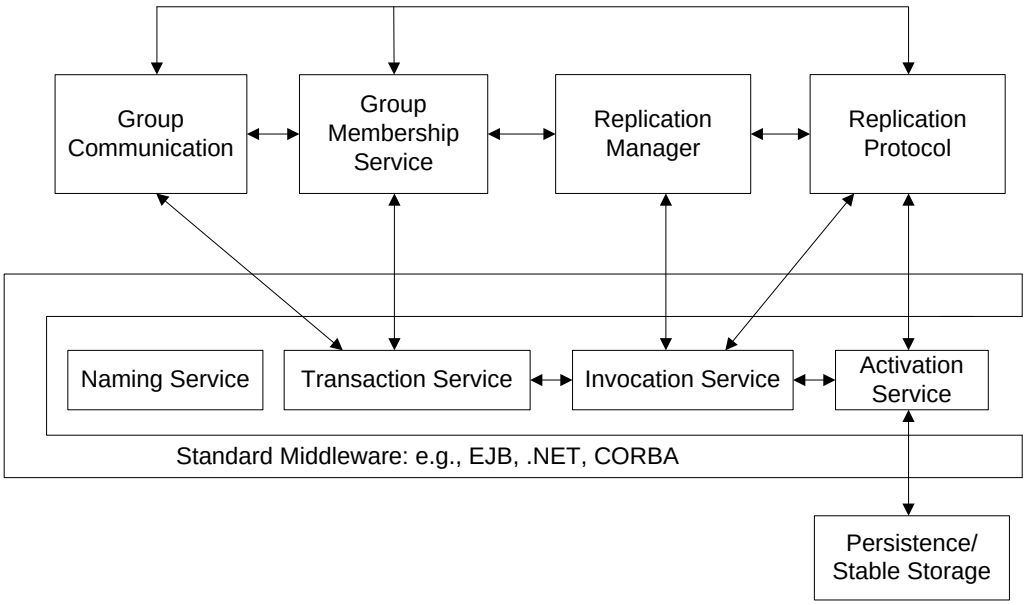

Figure 4.1: DeDiSys replication middleware

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 (e.g., primary or backup replica). The DeDiSys replication manager supports the primary-backup 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 (e.g., node or link) failures is also encapsulated 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 Adaptive Voting Protocol [3] or the Primary-Per-Partition Protocol [16] 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 [CKV01].

4.1.2 Fault-Tolerant CORBA

Originally, CORBA [OMG04], a popular middleware framework for object-oriented distributed systems, lacked of support for fault tolerance. Thus, FT-CORBA [OMG04] 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.

Figure 4.2 gives an overview on the FT-CORBA architecture. The architecture has been slightly redrawn and simplified from the FT-CORBA specification [OMG04]. The core components of the FT-CORBA infrastructure are the replication manager, fault notifier and detector, the logging, and the recovery mechanism. The FT-CORBA 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

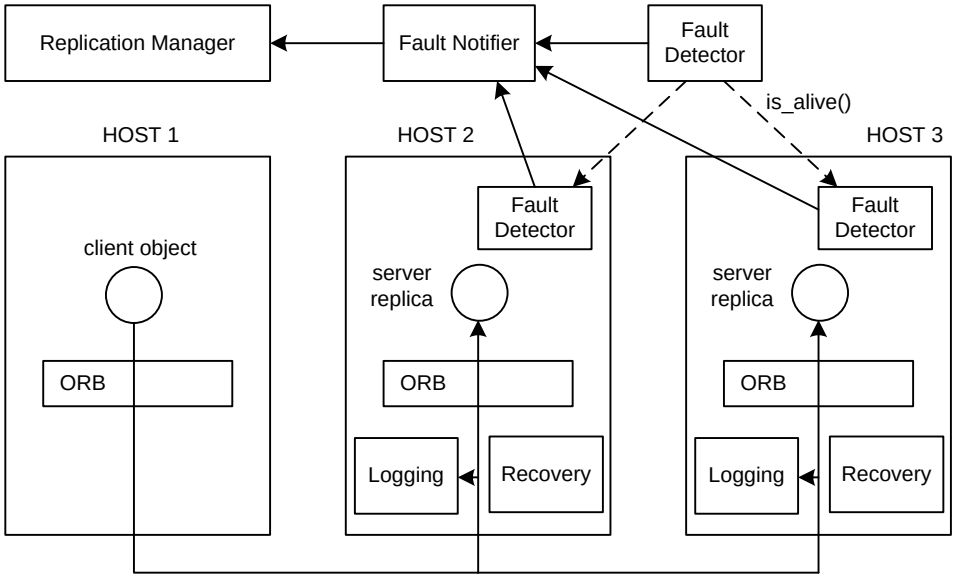

Figure 4.2: FT-CORBA architecture [OMG04]

manager. This allows the replication manager to act upon reported failures, 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 [BDLA04].

4.2 Replication Middleware for Service Oriented Systems

Few middleware solutions have been proposed for service oriented systems: The middleware systems presented in [YS05, SPPJ06] offer transparent active replication based on group communication [CKV01]. Primary-backup replication of Web services is offered by the FT-SOAP (Fault-Tolerant SOAP[1]) middleware [LFCL03]. Thema, a byzantine fault-tolerant middleware for Web service applications is proposed in [MIM+05]. The J2EE replication framework ADAPT [BBM+04], which is integrated into the JBoss [JBo] application server, allows to plug-in replication protocols and supports replication of Enterprise

[1]SOAP [W3C] is a messaging protocol used in Web service environments.

JavaBeans as well as Axis Web services. However, the latter 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 [LFCL03] and the active replication middleware systems [YS05, SPPJ06] 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 specification [OMG04] and the DeDiSys replication middleware [5] for distributed objects.

4.2.1 Primary-Backup Replication Middleware

Figure 4.3 shows the architecture of the FT-SOAP [LFCL03] replication middleware, which has many similarities with FT-CORBA [OMG04] and is based on the Apache Axis SOAP engine [Apab]. The key components in FT-SOAP are the *Replication Manager*, the *Fault Management* unit and the *Logging & Recovery* unit. SOAP engine interceptors, which are associated with each service, are used to intercept client requests. WSDL (Web Services Description Language [OAS]) files of replicated services are published in a UDDI (Universal Description, Discovery & Integration [OAS]) registry.

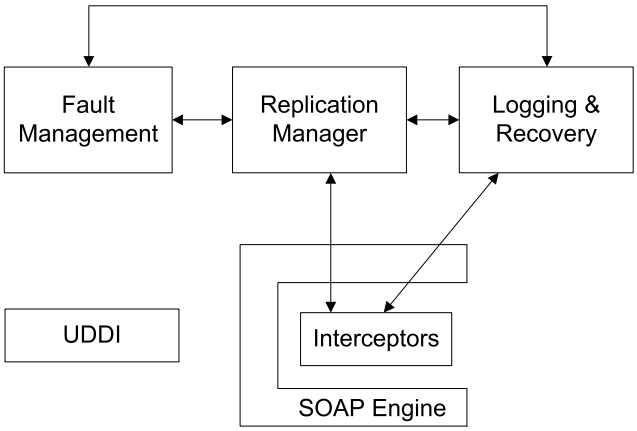

Figure 4.3: FT-SOAP (derived from [LFCL03])

The FT-SOAP replication manager basically provides interfaces for

- setting the desired replication properties like the replication style, initial number of replicas, etc. and
- creating and managing service groups.

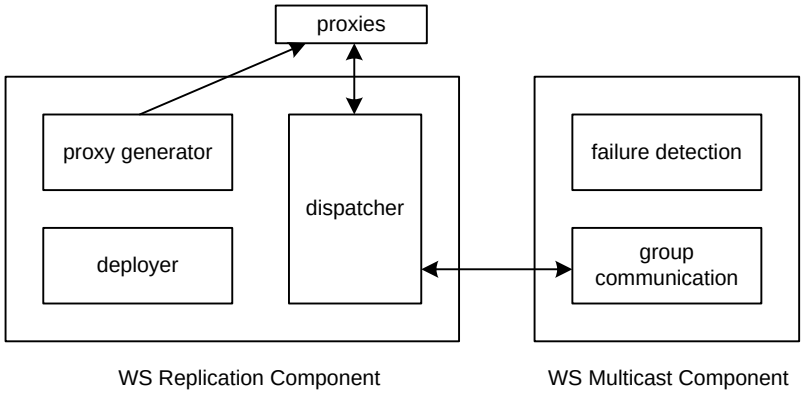

Figure 4.4: WS-Replication architecture (derived from [SPPJ06])

FT-SOAP extends WSDL by introducing a ⟨WSG/⟩ element in order to describe Web service groups.

The fault management unit, used to monitor replicated services, consists of a fault detector and a fault notifier. The fault detector actively polls Web services (according to a configurable polling frequency) in order to detect crashes and reports faults to a notification component, which in turn propagates the information to interested components, e.g., the replication manager.

Invocations and periodical checkpoints of the primary's state are logged in a database management system by the logging/recovery unit. Moreover, checkpoints are periodically transferred to the backups and logged invocations are replayed during the recovery stage if necessary.

4.2.2 Active Replication Middleware

The middleware presented by Ye and Shen [YS05] offers transparent active replication based on group communication. The system implements the TOPB-CAST [HB96] probabilistic multicast protocol using the JGroups toolkit [JGr]. Both synchronous and asynchronous interaction between the client and the Web service are supported.

WS-Replication [SPPJ06] is a framework for wide area replication of Web services and offers transparent active replication.

Figure 4.4 shows the architecture of the framework, which consists of two major components: 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. The Web service replication component is further structured into a Web service deployer, a dispatcher, and a proxy generator. Proxies are

used to intercept client invocations. The deployment facility enables the deployment of Web services (provided in a single archive) at all replicas. The dispatcher takes care of multicasting invocations using the WS-Multicast component. Besides multicast, 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 on the JGroups [JGr] toolkit.

4.3 Architectural Commonalities and Differences

Based on the representative architectures discussed in the previous sections, six major architectural units for both object and service replication middleware can be derived[2] as shown in Fig. 4.5: 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).

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.

4.3.1 Monitoring Service

Monitoring of the replicated entities is required both in distributed object and service oriented systems since replication middleware needs to take appropriate

[2]Of course, not every replication middleware physically strictly adheres to this separation of concerns. However, the functionality associated with the identified logical building blocks needs to be specified in both service and object replication middleware.

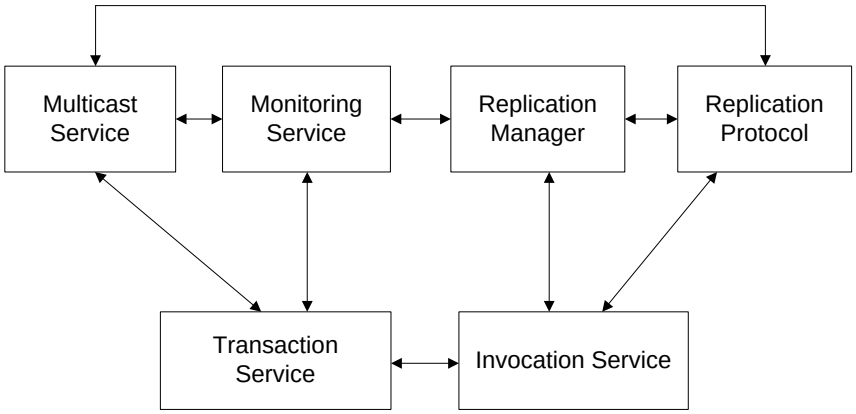

Figure 4.5: Generalized replication architecture

actions in case of failures of subparts (e.g., crash of a replica) in order to prevent system failures[3].

For instance, in case of primary-backup replication, it has to promote a backup to a new primary replica if the original primary crashes.

A monitoring service typically distinguishes three entities: *monitors* (failure detectors) collect information about failures of *monitorable entities* and disseminate this information to *notifiable entities*. In a push-model, monitorable entities periodically inform monitors about their status by using heartbeat messages. In a pull-model, monitors actively request liveness messages from monitorable entities. [FDGO99]

A special kind of monitoring service is a *group membership service* (GMS), which keeps track of membership changes of *dynamic* groups, caused by voluntary (join or leave) changes or failures (crashed or unreachable nodes). That is, a group membership service provides monitoring on the node level. In order to react to faults in a consistent way, 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" [Bir05b].

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

[3]A failure "is an event that occurs when the delivered service deviates from correct service." [ALRL04] Failures per se are hard to observe; however, their occurrence can be deduced by the detection of errors, which are deviations from correct system state. An error is caused by a fault. [ALRL04] It is important to note that system boundaries need to be taken into consideration when applying this taxonomy. For instance, with respect to replication middleware, a failure of a subsystem is considered a fault on the system level.

exist in *partitionable* membership services.

Vogels and Re defined a monitoring and membership service called WS-Membership [VR03] which is specifically targeted to Web services environments.

4.3.2 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 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 [CKV01]. Group communication systems provide multicast primitives to (object, process, service) groups with configurable delivery and ordering guarantees.

The following ordering guarantees can be distinguished:
FIFO multicast primitives guarantee that messages are received in FIFO order: If message m is sent before message m' by the sender, than every group member that receives both messages will receive m before m'.

Causal multicast primitives guarantee that messages are received in causal order: If message m causally precedes[4] [Lam78, Lam86] m', then every group member that receives both messages will receive m before m'.

Total order multicast primitives guarantee that messages are received in the same order at all correct group members. Defago et al. [DSU04] provide a comprehensive taxonomy and survey of total order protocols.

Reliability is defined in the context of view delivery, i.e., "reliability guarantees restrict message loss within a view" [CKV01]: *Sending view delivery* guarantees that a message is sent and delivered in the same view. *Same view delivery* is weaker and guarantees that a message is "delivered at the same view at every process that delivers it" [CKV01].

Schiper [Sch06] points out that group communication 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.

[4]Lamport called "precedes" initially "happened before" or "potentially causally dependent".

Indeed, most of the presented middleware architectures rely on group communication. Examples for state-of-the-art group communication toolkits are JGroups [JGr], Spread [ADS00], or the newly proposed SOAP-based WS-Multicast toolkit [SPPJ06], 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) [OAS] interface. However, this could also be realized for the other toolkits.

4.3.3 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 [LFCL03], DeDiSys [5], FT-CORBA [OMG04]). The *Web service replication component* of the WS-Replication framework [SPPJ06] provides similar functionality.

For instance[5], the interface of a simple primary-backup replication manager might contain the following methods:

- `getPrimary()`: returns the location of the primary replica
- `getBackups()`: returns the locations of the backup replicas
- `getReplica(replicaID)`: returns the location of a replica with the given identifier
- `addReplica(location,role)`: adds a new replica with its location and role
- `changeRole(replicaID,role)`: changes the role of a replica
- `deleteReplica(replicaID)`: deletes a replica

A replication manager for quorum consensus protocols does not distinguish between primary and backup replicas, but requires methods for retrieving read and write quorums of replicas.

Though the tasks of the replication manager 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

[5]The DeDiSys replication manager has a similar interface, see [Fro05].

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, can influence the choice of the data structure for the replica location service which is part of the replication manager.

4.3.4 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 [MMN98] or in the WS-Replication middleware [SPPJ06]). 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.

4.3.5 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 [5] uses the interception mechanism of the JBOSS application server (in the J2EE version of the framework), .NET Remoting interceptors (in the .NET variant), and CORBA portable interceptors (in the CORBA variant). The Web service replication component of the WS-Replication framework [SPPJ06] 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.

The SOAP engine Axis2 [Apac, PHE+06] allows the definition of customizable message interceptors, so-called "handlers", which ease the implementation of an invocation service in a Java-based Web service environment. Microsoft's new SOAP-based communication platform Windows Communication Foundation (WCF [Micd]), which is part of the .NET Framework 3.0 [Micc], also allows the interception of message calls.

	Object Replication Middleware	Service Replication Middleware
Granularity	typic. *fine-grained* (objects)	typic. *coarse-grained* (services)
Multicast S.	GC beneficial	GC beneficial
Monitoring S.	failure detector, GMS	failure detector, GMS
Transaction S.	typic. ACID trans.	ACID and *long running trans.*
Replication M.	maintains *large* nr. of objects	maintains *small* nr. of services
Replication P.	separate or embedded	separate or embedded
Invocation S.	interceptors	interceptors

Table 4.1: Commonalities and differences of replication middleware systems

4.3.6 Transaction Service

Transactions are a fault tolerance technique by themselves; specifically the atomicity and durability properties of traditional ACID[6] transactions are related to fault tolerance [Sch06]. 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 [5]), support for transactions in replication middleware for service oriented systems is rather in its infancy. WS-Replication [SPPJ06] has been successfully tested in combination with long running transactions as defined in the Web Services Composite Application Framework (WS-CAF) [OAS]. 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 4.3.6 summarizes the commonalities and differences of object and service replication middleware.

4.3.7 Interactions Between Components

So far only the (logical) static structure of object and service replication middleware has been discussed; however, dynamic behavior needs to be considered as well. Thus, figure 4.6 exemplary[7] shows the interaction of the main components

[6] atomicity, consistency, isolation, durability

[7] Of course, this does not mean that every primary-backup replication middleware strictly follows this call flow. However, this call flow is feasible as has been shown by our proof-of-

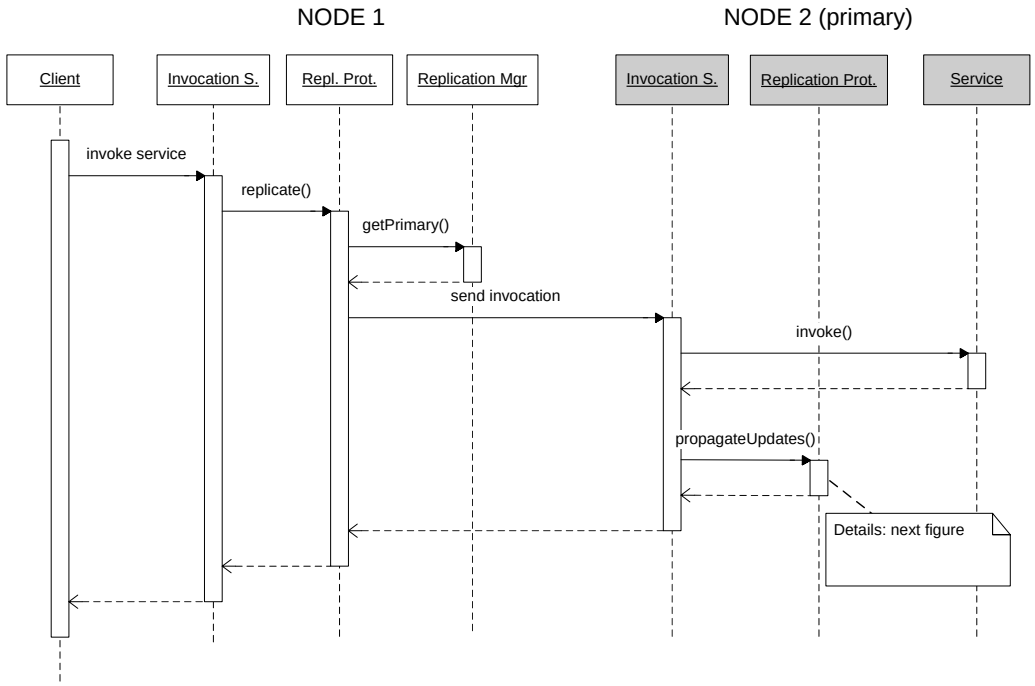

Figure 4.6: Interactions between key components

of the derived architecture for primary-backup replication: Client invocations are intercepted by the invocation service in order to trigger the replication logic. First, the replication protocol requests the location of the primary copy from the replication manager. If it does not reside on the current node, the invocation is redirected to the node hosting the primary. The invocation is processed by the primary and afterwards the update is propagated (either via state transfer or operation transfer) to the backup replicas.

Update propagation for primary-backup replication is depicted in Figure 4.7. The replication protocol queries the replication manager to get the location (or group ID) of the backup replicas and then uses the multicast service to propagate the updates to the backup replicas, which apply the update.

concept implementations for both service [9] [Weg07] and object replication middleware [5].

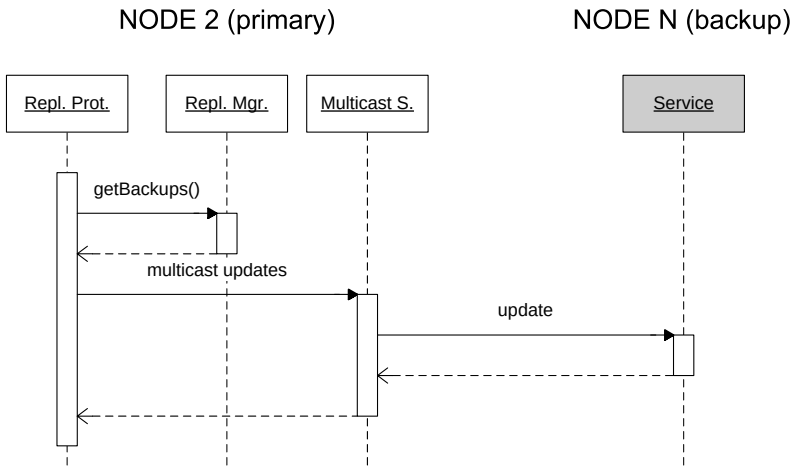

Figure 4.7: Update propagation

4.4 Proof of Concept Implementations: Lessons Learned

Besides the DeDiSys replication middleware for distributed objects described in section 4.1.1, a Web service replication middleware built-upon the Java-based Axis2 SOAP engine [PHE+06] has been implemented [Weg07], [9]. The middleware closely follows the architectural pattern presented in the previous section and provides a variant of primary-backup replication using operation transfer. The entry point to the replication system is the invocation system which is based on the Axis2 handler concept. Axis2 handlers are customizable message interceptors. Incoming SOAP messages are cut out of the Axis IN-flow, propagated to the other replicas and injected in their IN-flow. Monitoring and multicast primitives are provided by the Spread [ADS00] group communication toolkit. Performance evaluations of the middleware show the relatively low overhead of replication in a local setting if the number of replicas is small. The overhead is primarily caused by the membership management and multicast algorithms used by Spread. Therefore, scalability of our middleware is similar to the scalability of Spread. Of course, the overhead increases in a wide area setting due to increased network latency. Details about our Axis2-based replication middleware can be found in [9] and [Weg07].

The main lessons learned from design and implementation of the distributed object (DeDiSys EJB, .NET, CORBA variants) and Web service replication middleware prototypes (Axis2-based framework) are summarized in the follow-

ing sections:

4.4.1 Key Component: Group Communication Toolkit

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.

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

4.4.2 Invocation Service

.NET: .NET Remoting, which is used in the .NET version of the DeDiSys replication middleware, allows to inject custom proxies (extensions of RealProxy) for interception of invocations on the client and server side of the invocation chain. Thus, the .NET framework provides sufficient support for realizing an invocation service as used in our middleware.

CORBA: The CORBA-based DeDiSys replication middleware [BMG06] is based on JacORB [Jac] and uses CORBA portable interceptors [OMG04] 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 [JBo]. 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 [Apac, PHE+06] 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. [9] [Weg07]

Lesson: 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 integrate replication logic and achieve replication transparency.

4.4.3 Separation of Replication Management and Protocol

.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. [6]

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 [BMG06].

J2EE: For the J2EE DeDiSys replication service, we build upon the replication framework of ADAPT [WKM04]. 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[8] 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.

Lesson: 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.

4.4.4 Transaction Support

.NET: Transaction support has not been implemented in the .NET version of the DeDiSys middleware. Nevertheless, transactional support could for instance be based on the Microsoft Distributed Transaction Coordinator.

CORBA: The CORBA-based DeDiSys transaction manager adheres to the Java Transaction API (JTA) [Sun] but has been implemented from scratch since network failures are often not properly treated by off-the-shelf transaction managers (see [Zag07]).

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 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 [OAS]) provides a foundation layer for consensus between Web services, where specific consensus protocols can be built upon, e.g., distributed transactions.

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

Two particular specifications for Web service transactions build upon the WS-Coordination framework: WS-AtomicTransaction [OAS] for short running ACID transactions and WS-BusinessActivity [OAS] for long running transactions with weaker guarantees.

Apache Kandula [Apaa] 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: Combining replication with transactions is a major challenge for service replication middleware.

4.5 Summary

Replication protocols like Adaptive Voting (see chapter 3) that follow the Availability/Consistency Balancing Replication Model (ACBRM, see chapter 2) require extensions to standard replication middleware (e.g., FT-CORBA [OMG04]). The DeDiSys replication middleware, which is targeted to distributed object systems, provides such extensions for CORBA, .NET, and EJB technologies. In this chapter, the DeDiSys middleware and FT-CORBA — two examples for distributed object replication middleware architectures — have been compared with replication middleware for service oriented systems on an architectural level. The comparison was focused on middleware systems that provide well-established replication protocols such as primary-backup or active replication or variants thereof.

The analysis shows the commonalities (and only subtle differences) on an architectural level: Both service and object replication middleware architectures can be (at least logically) divided into six infrastructure components: an invocation service, a replication manager, a replication protocol, a monitoring service, a multicast service, and an optional transaction service. Based on this analysis and experiences with our own middleware implementations we conclude that many well-established replication middleware architectures can be reapplied in service oriented systems.

Chapter 5

Related Work, Conclusions, and Future Work

This chapter provides a insight into related work, provides a summary of the main contributions, the associated conclusions, and gives an outlook to future work.

5.1 Related Work

Subsection 5.1.1 covers closely related work in the area of optimistic replication. Reconciliation strategies are depicted in subsection 5.1.2. The balancing of quality of service with availability, which has been investigated in the DeDiSys project as well, is discussed in subsection 5.1.3. Related work on replication in service oriented systems is discussed in subsection 5.1.4. Related work on replication middleware architectures has already been covered in chapter 4. Related work on comparisons of service and object replication middleware is finally discussed in section 5.1.5.

5.1.1 Optimistic Replication in Data-Centric Systems

A detailed survey on optimistic replication techniques can be found in [SS05] but not with a focus on the balancing between data integrity and availability.

Trading replica consistency for increased availability has been addressed in distributed object systems as [FN02, GFGM98, RBC+03]. However, these systems either guarantee strong replica consistency or no replica consistency at all. TACT (Tunable Availability and Consistency Trade-offs) [YV02] fills the space in between by providing a continuous consistency model based on logical consistency units (*conits*). The consistency level of each conit is defined using three application-independent metrics — numerical error, order error,

and staleness. For instance, in a replicated bulletin board service, where users can post messages to any replica or retrieve messages from any replica, a conit covers all news messages. Numerical error limits the total number of messages posted system-wide but not seen locally, order error bounds the number of out-of-order messages on a given replica, and staleness limits the delay of messages. The TACT replication system enforces that the specified limits are not exceeded. TACT provides a fine-grained trade-off between replica consistency and availability but does not focus on constraint consistency.

In [FM04], the application developer can define replica consistency on *Data Objects* using a large set of parameters. Data objects are passive entities that encapsulate data and provide operations on the data but do not — in contrast to objects in our target applications — invoke other objects.

While our approach treats disconnected operation as a failure scenario, disconnections are inherent in mobile environments. Thus, different solutions for reconciliation of divergent, conflicting replicas have been proposed for mobile environments: In Bayou [TTP+95], application developers need to define application-specific conflict detection and reconciliation policies. Replica consistency is re-established by an anti-entropy protocol with eventual consistency guarantees. Our approach offers the same flexibility as Bayou but offers predefined reconciliation policies in addition. Gray et al. [GHOS96] introduced the concept of tentative transactions: Transactions are tentatively committed on replicated data on mobile (disconnected) nodes and later applied at a master copy when the nodes rejoin. If the commit on the master copy fails, the originating node is informed why it failed. Application-specific semantics are used for conflict resolution in the mobile transaction management system presented in [PBM+00]. Our solution is also applicable in non-transactional applications.

Beside the already mentioned differences to our approach, *all* of the above replication and reconciliation approaches have one commonality: In contrast to our approach, they either do not address constraint consistency explicitly or presume strict constraint consistency.

5.1.2 Reconciliation Strategies

Asplund's work [AN06a, AN06b, ANBG07] is focused on reconciliation protocols for the Primary-Per-Partition-Protocol. In principle, the proposed techniques can also be used for the Adaptive Voting replication protocol presented in chapter 4 of this thesis. Thus, Asplund's work complements this thesis since concrete reconciliation protocols were out of the scope of this thesis.

In [AN06a], Asplund et al. propose three different fully automatic reconciliation algorithms. The authors assume that no failures occur during reconcili-

ation. Updates are not accepted during reconciliation in these approaches.

The `CHOOSE1` algorithm chooses the partition with the highest utility from a set of constraint consistent partitions. Each operation is assigned a certain utility value representing the usefulness of the operation for an application. Thus, the operations applied during degraded mode need not be logged for this reconciliation algorithm but the utility of each partition needs to be determined by some means. If all operations have the same utility, the partition with the most updates during degradation is chosen.

The algorithms `MERGE` and `Greatest Expected Utility (GEU)` perform operation-based reconciliation. The non-deterministic MERGE algorithm replays operations (in an arbitrary order) tentatively applied during degraded mode. Operations are only applied if they do not violate constraint consistency; otherwise they are rejected. The GEU algorithm advances the MERGE algorithm by ordering the operations based on the highest expected utility in order to maximize the expected utility. The expected utility is a measure that takes the actual utility of an operation and the probability of violation — i.e., the probability that the operation will violate data integrity — into account.

In [AN06b, ANBG07], Asplund et al. propose a reconciliation algorithm that services operations during reconciliation by maintaining virtual partitions. As MERGE and GEU, this algorithm is also operation-based but — in contrast to the above ones — takes the ordering of operations invoked by the same client into account during replay.

5.1.3 Balancing Quality of Service and Availability

In the DeDiSys (Dependable Distributed Systems) project, we have addressed fault tolerance techniques for both data-centric and resource-centric services. Replication protocols and middleware for the former kind of systems are the focus of this thesis.

Resource-centric systems are mainly concerned about usage or management of resources (e.g., processing power, storage, network bandwidth etc.). Different aspects such as time, capacity, protocols, or resource allocation come into consideration here. For this kind of systems, DeDiSys focused on the trade-off between availability and quality of service (QoS) and introduced a system for balancing these attributes.

More specifically, a middleware architecture has been developed that automatically matches provided and requested service capabilities taking the minimal requirements of consumers into consideration as well in order to enhance availability. Details on the key concept of the trading approach and the middleware architecture are presented in [Kar06, Moc07]. On a very high level, this approach is similar to the trading concept pursued in this thesis: availabil-

ity is balanced against consistency. While consistency refers to the satisfaction of data integrity constraints in the data-centric approach, consistency of requested and provided capabilities is subject to trading in the resource-centric approach. More details on a comparison between the two approaches can be found in [Kar07].

5.1.4 Replication in Service Oriented Systems

Literature on replication techniques for service oriented systems has already been extensively cited in previous chapters. Here, references to other research on the application of fault tolerance techniques in service oriented systems are summarized.

Moser et al. [MMZ06] discuss fault tolerance techniques for Web services including replication, checkpointing and message logging. However, they do not focus on replication of composite services. Nevertheless, their paper complements our work by discussing which infrastructure components need to be replicated in a service oriented system to achieve dependability: e.g., the UDDI registry, the transaction coordinator, etc.

Birman [Bir06] stresses the importance of replication in service oriented mission-critical systems in order to achieve robustness and trustworthiness. The same author suggests using the virtual synchrony process group computing model and the state machine model (e.g., exemplified by Lamport's Paxos [Lam98] algorithm) to deploy replication in Web services based systems [Bir05a]. Indeed, some of the replication methods for stateful services that we have discussed require multicast primitives [GS97] which could be provided by toolkits (targeted to service oriented systems) that follow the virtual synchrony model, for example.

Dependability of composite Web services with online upgrades has been discussed by Gorbenko et al. [GKPR05], but replication is only indirectly addressed with respect to the parallel execution of several releases of a Web service. Other dependability mechanisms for composite services (but not replication) such as backward or forward error recovery are for example addressed by Tartanoglu et al. [TIRL03].

Besides conceptual considerations regarding replication in the above mentioned publications, some literature exists on the implementation of specific replication techniques in service oriented systems as discussed in the previous section.

Replication of data stores such as databases or file systems has been extensively discussed in scientific literature. Oliveira et al. give an excellent overview about state-of-the-art in database replication [Gor05]. Saito and Shapiro [SS05] provide a survey on optimistic replication techniques (weak consistency guar-

antees) including both file and database replication techniques. Wiesmann et al. [WPS+00] compare replication in database systems with replication in distributed object or process systems. However, combinations of these approaches with replication on the service layer have not been investigated to our knowledge.

5.1.5 Comparisons of Object and Service Replication Middleware

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 [SPPJ06] but not described. Similarities between FT-SOAP [LFCL03] and FT-CORBA [OMG04] are mentioned in [LFCL03], but they do not cover other middleware solutions. Although object replication middleware is well-established, architectural comparisons are even hard to find within this area. One noteworthy exception [FN04] 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., [RDH05]) 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 chapter 4, 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.

5.2 Summary and Conclusions

In this section, the three main contributions of this thesis — the Availability/Consistency Balancing Replication Model, the Adaptive Voting Replication Protocol, and the comparison of replication middleware architectures — are summarized and the main conclusions highlighted.

5.2.1 Availability/Consistency Balancing Replication Model

Chapter 2 presented the Availability/Consistency Balancing Replication Model (ACBRM), an enhancement of Wiesmann et al's replication model [WPS+00] that enables the balancing of data integrity with availability in degraded situations when node and link failures occur. The key idea is to allow non-critical operations in degraded situations in all partitions even if replica conflicts arise and data integrity constraints are possibly violated (threatened). This requires reconciliation policies to re-establish replica and constraint consistency when nodes recover and network partitions rejoin.

The Primary-per-Partition-Protocol [16] and Adaptive Voting [3] are two concrete protocols that follow the ACBRM. The feasibility of our approach has been shown by several prototype implementations (e.g., [15, 4], [BMG06]).

5.2.2 Adaptive Voting

Chapter 3 introduced Adaptive Voting [3], an enhanced replication protocol based on traditional voting, that allows to balance data integrity against availability in degraded situations when node and link failures occur. The key idea of Adaptive Voting is to allow non-critical operations (that only affect tradeable data integrity constraints) even if the quorum conditions cannot be met. Different reconciliation policies can be used to re-establish consistency during reconciliation.

The availability analysis presented in this thesis shows that Adaptive Voting provides better availability than traditional voting if (i) a significant portion of the data integrity constraints can be temporarily relaxed and (ii) reconciliation time is shorter than degradation time. Performance measurements [4] [Chl07] give an indication for the choice of the quorum strategy in terms of performance. As in traditional voting, a read-one/write-all strategy is best for read-intensive applications while a majority strategy is better for write-intensive applications. However, implicit read operations caused by constraint checking during a write operation (e.g., in case of inter-object constraints) have to be taken into account.

5.2.3 Replication Middleware Architectures

In chapter 4, state-of-the-art service replication middleware has been compared with object replication middleware on an architectural level. The focus was on the replication middleware for the most common replication techniques, namely active and primary-backup replication.

5.2. SUMMARY AND CONCLUSIONS

Replication is rather in its infancy in service oriented systems; thus, only a small number of replication middleware solutions was suitable for our comparison [SPPJ06, LFCL03, YS05].

Among the replication architectures for distributed object systems we have chosen the well-known FT-CORBA standard [OMG04] and the DeDiSys architecture [5] 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 can be divided into six major infrastructure components:

- A *Multicast Service* for reliable, ordered dissemination of operations.
- A *Monitoring Service* for detection of failures 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 (such as primary-backup or active replication) are needed that allow for (but are not limited to) strict consistency, 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 reapply the same architectures in service oriented systems.

5.3 Future Work

An obvious starting point for future work is to apply the concept of balancing data integrity with availability on the database layer instead on the object layer. On the technological side, future work is to enhance the Axis2-based replication middleware for Web services, e.g., with transactional support. Standardization of Web service replication techniques is another worthwhile goal and last but not least additional challenges in service oriented environments such as cross-organizational heterogeneity, massive scale, dynamics could be addressed in future projects.

5.3.1 Balancing Data Integrity with Availability on the Database Layer

The current solution (shown in Fig. 5.1) of the DeDiSys middleware for balancing data integrity with availability performs replication on the object level using the EJB, .NET, or CORBA variant of our distributed object replication middleware. Nevertheless, data objects (e.g., Entity beans in EJB terminology) are persisted in a (relational) database in this architecture.

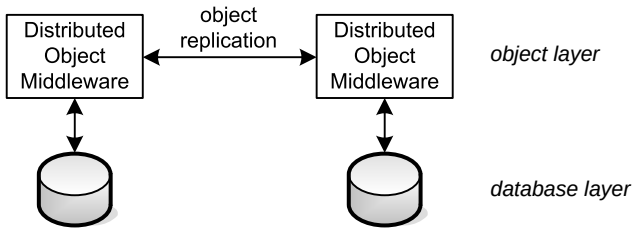

Figure 5.1: Replication on the object layer

Future work should investigate, if the DeDiSys approach could also be — at least partly — implemented on the data layer. That is, replication could also be performed on the data layer instead on the distributed object layer. Figure 5.2 shows the revised layering in this case.

Both commercial and open source state-of-the-art database management systems such as Oracle, IBM DB2, Microsoft SQL Server 2005, PostgreSQL, or MySQL support replication. That is, the wheel would not have to be re-invented from scratch but existing database replication frameworks (primarily for open source databases) would have to be extended for realizing the DeDiSys approach on the database layer. One advantage of this architecture could be

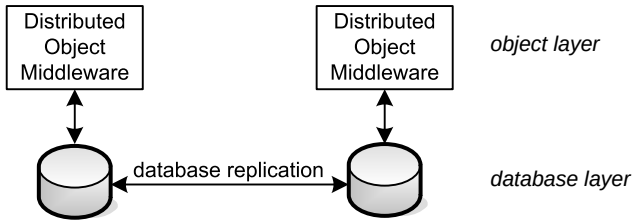

Figure 5.2: Replication on the database layer

the inherent support for data integrity constraint checking by database management systems. Primary keys, unique keys, foreign keys, not null constraints, and check clauses (e.g., for restricting values of an attribute) can be described in a declarative way by using SQL (Structured Query Language) and are automatically enforced by the database management system. However, more complex constraints would require advanced concepts such as triggers. Furthermore, it is subject to future investigation how the trading can be performed. Static negotiation, that is the application of predefined rules on which constraints can be temporarily relaxed during degradation, could probably be implemented using a database stored procedure language. However, in case of dynamic negotiation which requires application interaction (and possibly user interaction), at least parts of the negotiation mechanism would have to be implemented on the object layer. To summarize, future research should show if the DeDiSys approach is feasible on the database layer and compare it with the current distributed object solution.

5.3.2 Enhancements of the Axis2-based Web Service Replication Middleware

The Web service replication middleware [9] based on the Axis2-SOAP-engine, does currently not support transactions and does not take security issues into consideration. Future work on this framework should focus on the combination of Web service replication mechanisms with Web service transaction concepts such WS-AtomicTransaction [OAS] and WS-BusinessActivity [OAS]. Moreover, other replication protocols than the currently implemented primary backup variant such as an update everywhere approach could be implemented and evaluated. Finally, a security concept for the replication middleware should be investigated as well if it shall be applied in security critical application areas such as banking and finance.

5.3.3 Standardization of Web Service Replication

So far, no standard/specification for Web service replication exists: The existing Web service replication solutions — including our framework [9] — build upon basic Web service standards such as SOAP [W3C] and WSDL [W3C] but do not define a "WS-Replication standard"[1]. A reason for the lack of a common standard/specification for Web service replication is that Web service replication is currently mainly an intra-enterprise concern. That is, replication is not applied across organizational boundaries and thus a standard would not be as beneficial as for example for distributed transactions 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 [9] 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 argued in chapter 4. Some differences compared to traditional replication systems are caused by the coarse-grained nature of services, which allows certain optimizations (e.g., internal data structures) in the service replication middleware. Scalability, another major challenge addressed by service oriented architectures, can be mastered by looser forms of replication coupling. For instance, in case of a primary-backup scheme, loose replication coupling can be achieved by asynchronous update propagation.

While state-of-the-art Web service replication middleware frameworks (including our solution) address 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 [FGG+06] 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 [OAS] — for replication protocols, group membership services, group communication protocols, etc. are likely to be necessary for such settings.

5.3.4 Challenges for Dependability of Service Oriented Systems of the Future

While some traditional dependability concepts (such as replication as mentioned above) can be adopted and adapted for today's service oriented sys-

[1] Salas et al. [SPPJ06] call their replication solution "WS-Replication". However, it is only a name for their framework and they do not suggest a standard or specification.

tems, additional dependability research is required for systems of the future characterized by cross-organizational heterogeneity, massive scale, and dynamics. A plethora of research questions needs to be answered, for instance: What does dependability (and security) mean in heterogeneous, massive scale, dynamic systems? How can dependability/security properties be balanced against each other? What are the user needs regarding dependability and security in these systems? What are the constraints for dependability and security in such settings? How can dependability and security concepts be integrated? In addition, existing dependability and security protocols need to be assessed regarding their suitability for service oriented environments. Based on these analyses, novel (or extensions of existing) protocols and services for discovery, group membership, replication, transactions, etc. need to be defined to cover the requirements not addressed by today's techniques.

List of Figures

LIST OF FIGURES

List of Tables

Bibliography

[AD76] Peter A. Alsberg and John D. Day. A principle for resilient sharing of distributed resources. In *ICSE '76: Proc. 2nd Int. Conf. on Software engineering*, pages 562–570. IEEE CS Press, 1976.

[ADS00] Yair Amir, Claudiu Danilov, and Jonathan Stanton. A low latency, loss tolerant architecture and protocol for wide area group communication. In *Proc. of The Int. Conf. on Dependable Systems and Networks*, pages 327–336. IEEE, 2000.

[ALRL04] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.*, 1(1):11–33, 2004.

[AN06a] Mikael Asplund and Simin Nadjm-Tehrani. Post-partition reconciliation protocols for maintaining consistency. In *Proc. of the 21st ACM Symp. on Applied Computing*, pages 710–717. ACM Press, 2006.

[AN06b] Mikael Asplund and Simin Nadjm-Tehrani. *Rigorous Development of Complex Fault-Tolerant Systems*, volume 4157 of *Lecture Notes in Computer Science*, chapter Formalising Reconciliation in Partitionable Networks with Distributed Services, pages 37–58. Springer Verlag, 2006.

[ANBG07] Mikael Asplund, Simin Nadjm-Therani, Stefan Beyer, and Pablo Galdamez. Measuring availability in optimistic partition-tolerant systems with data constraints. In *Proc. of the 37th Int. Conf. on Dependable Systems and Networks*. IEEE CS, 2007.

[Apaa] Apache. Apache Kandula2, http://ws.apache.org/kandula/2/.

[Apab] Apache. Axis, http://ws.apache.org/axis/.

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

[BBM+04] Özalp Babaoglu, Alberto Bartoli, Vance Maverick, Simon Patarin, Jaksa Vuckovic, and Huaigu Wu. A framework for prototyping J2EE replication algorithms. In *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, volume 3291 of *LNCS*, pages 1413–1426. Springer, Jan. 2004.

BIBLIOGRAPHY

[BDLA04] Alysson Neves Bessani, Joni Da Silva Fraga, Lau Cheuk Lung, and Eduardo Adílio Pelinson Alchieri. Active replication in CORBA: Standards, protocols, and implementation framework. In *Proc. of CoopIS/DOA/ODBASE (2)*, pages 1395–1412, 2004.

[Bir05a] Ken Birman. Can web services scale up? *IEEE Computer*, 38(10):107–110, 2005.

[Bir05b] Ken Birman. *Reliable Distributed Systems*. Springer, 2005.

[Bir06] Ken Birman. The untrustworthy web services revolution. *IEEE Computer*, 39(2):98–100, 2006.

[BJRA85] Ken Birman, Thomas A. Joseph, Thomas Raeuchle, and Amr El Abbadi. Implementing fault-tolerant distributed objects. *IEEE Trans. on Software Engineering*, 11(6):502–508, 1985.

[BMG06] Stefan Beyer, Francesc Muñoz-Escoi, and Pablo 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.

[BMST93] Navin Budhiraja, Keith Marzullo, Fred B. Schneider, and Sam Toueg. The primary-backup approach. In S.J. Mullender, editor, *Distributed systems*, chapter 8, pages 199–216. ACM Press, Addison-Wesley, 2nd edition, 1993.

[Chl07] Norbert Chlaupek. Implementation and evaluation of the adaptive voting replication protocol in a .net environment. Master's thesis, University of Applied Sciences fh-campus wien, 2007. Advisors: Johannes Osrael and Lorenz Froihofer.

[CKV01] Gregory Chockler, Idit Keidar, and Roman Vitenberg. Group communication specifications: a comprehensive study. *ACM Computing Surveys*, 33(4):427–469, 2001.

[Con] Continuent. http://www.continuent.com.

[Cri91] Flaviu Cristian. Understanding fault-tolerant distributed systems. *Commun. ACM*, 34(2), 1991.

[Dav84] Susan B. Davidson. Optimism and consistency in partitioned distributed database systems. *ACM Trans. Database Syst.*, 9(3):456–481, 1984.

[DGH+87] Alan Demers, Daniel Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Daniel Swinehart, and Douglas Terry. Epidemic algorithms for replicated database maintenance. In *PODC '87: Proc. of the sixth annual ACM Symp. on Principles of distributed computing*, 1987.

BIBLIOGRAPHY

[DSU04] Xavier Defago, André Schiper, and Peter Urban. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Computing Surveys*, 36(4):372–421, 2004.

[FDGO99] Pascal Felber, Xavier Défago, Rachid Guerraoui, and Philipp Oser. Failure detectors as first class objects. In *Proc. Int. Symp. on Distributed Objects and Applications*, pages 132–141. IEEE CS, 1999.

[FFT+04] Ian Foster, Jeffrey Frey, Steve Tuecke, Karl Czajkowski, Don Ferguson, Farnk Leymann, Martin Nally, Igor Sedukhin, David Snelling, Tony Storey, William Vambenepe, and Sanjiva Weerawarana. Modeling stateful resources with web services, 2004.

[FGG+06] Peter Feiler, Richard Gabriel, John Goodenough, Rick Linger, Tom Longstaff, Rick Kazman, Mark Klein, Linda Northrop, Douglas Schmidt, Kevin Sullivan, and Kurt Wallnau. *Ultra-Large-Scale Systems*. Software Engineering Institute Carnegie Mellon, 2006.

[FLCL04] Chen-Liang Fang, Deron Liang, Chyouhwa Chen, and PuSan Lin. A redundant nested invocation suppression mechanism for active replication fault-tolerant web service. In *Proc. Int. Conf. on e-Technology, e-Commerce and e-Service*, pages 9–16. IEEE CS, 2004.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.

[FM04] Corina Ferdean and Mesaac Makpangou. A generic and flexible model for replica consistency management. In *Proc. of the 1st Int. Conf. on Distributed Computing and Internet Technology*, LNCS 1884, pages 204–209. Springer Verlag, 2004.

[FN02] Pascal Felber and Priya Narasimhan. Reconciling replication and transactions for the end-to-end reliability of corba applications. In *Proc. of Confederated Int'l Conf. DOA, CoopIS and ODBASE 2002*, volume 2519 of *LNCS*, pages 737–754. Springer, 2002.

[FN04] Pascal Felber and Priya Narasimhan. Experiences, strategies, and challenges in building fault-tolerant CORBA systems. *IEEE Trans. Comput.*, 53(5):497–511, 2004.

[FP98] Jean-Charles Fabre and Tanguy Pérennou. A metaobject architecture for fault-tolerant distributed systems: The FRIENDS approach. *IEEE Trans. Comput.*, 47(1):78–95, 1998.

[Fro05] Lorenz Froihofer (ed.). Ftns system model. Technical Report D2.2.1, DeDiSys Consortium (www.dedisys.org), 2005.

BIBLIOGRAPHY

[Fro07] Lorenz Froihofer. *Middleware Support for Adaptive Dependability through Explicit Runtime Integrity Constraints*. PhD thesis, Vienna University of Technology, Austria, 2007.

[GFGM98] Rachid Guerraoui, Pascal Felber, Benoit Garbinato, and Karim Mazouni. System support for object groups. In *OOPSLA '98: Proc. of the 13th ACM SIGPLAN Conf. on Object-oriented programming, systems, languages, and applications*, pages 244–258. ACM Press, 1998.

[GHOS96] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. In *Proc. Int. Conf. on Management of Data*, pages 173–182. ACM, 1996.

[Gif79] David K. Gifford. Weighted voting for replicated data. In *SOSP '79: Proc. of the 7th ACM Symp. on Operating Systems Principles*, pages 150–162. ACM Press, 1979.

[GKPR05] Anatoliy Gorbenko, Vyacheslav Kharchenko, Peter Popov, and Alexander Romanovsky. Dependable composite web services with components upgraded online. In *Architecting Dependable Systems III*, volume 3549 of *LNCS*, pages 92–121. Springer, 2005.

[Gor05] Gorda Consortium. D1.1 - state of the art in database replication. Technical report, Universidade do Minho, Braga, Portugal, 2005.

[GS97] Rachid Guerraoui and André Schiper. Software-based replication for fault tolerance. *Computer*, 30(4):68–74, 1997.

[Hab07] Igor Habjan (ed.). Software Prototype and Refined Design. Technical Report D3.3.2, DeDiSys Consortium (www.dedisys.org), 2007.

[HB96] Mark Hayden and Ken Birman. Probabilistic broadcast. Technical report, Cornell University, Ithaca, NY, USA, 1996.

[HHB96] Abdelsalam A. Helal, Abdelsalam A. Heddaya, and Bharat B. Bhargava. *Replication Techniques in Distributed Systems*. Kluwer Academic Publishers, 1996.

[IBM] IBM. IBM DB2 Universal Database,. http://www.ibm.com/db2/.

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

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

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

[JPAK03] Ricardo Jiménez-Peris, Marta Patiño-Martínez, Gustavo Alonso, and Bettina Kemme. Are quorums an alternative for data replication? *ACM Trans. Database Syst.*, 28(3):257–294, 2003.

BIBLIOGRAPHY

[JPR+83] Douglas S. Parker Jr., Gerald J. Popek, Gerard Rudisin, Allen Stoughton, Bruce J. Walker, Evelyn Walton, Johanna M. Chow, David A. Edwards, Stephen Kiser, and Charles S. Kline. Detection of mutual inconsistency in distributed systems. *IEEE Trans. Software Eng.*, 9(3):240–247, 1983.

[KA00] Bettina Kemme and Gustavo Alonso. A new approach to developing and implementing eager database replication protocols. *ACM Trans. Database Syst.*, 25(3):333–379, 2000.

[Kar06] Piotr Karwaczynski (ed.). System Model Group Communication. Technical Report D2.1.1, DeDiSys Consortium (www.dedisys.org), 2006.

[Kar07] Piotr Karwaczynski (ed.). Theory Comparison. Technical Report D4.2.1, DeDiSys Consortium (www.dedisys.org), 2007.

[KRSD01] Anne-Marie Kermarrec, Antony Rowstron, Marc Shapiro, and Peter Druschel. The icecube approach to the reconciliation of divergent replicas. In *PODC '01: Proc. 20th ACM Symp. on Principles of Distributed Computing*, pages 210–218, New York, NY, USA, 2001. ACM Press.

[Kue07] Hubert Kuenig (ed.). Software Prototype and Refined Design and Validation Report. Technical Report D3.2.2, DeDiSys Consortium (www.dedisys.org), 2007.

[Lam78] Leslie Lamport. The implementation of reliable distributed multiprocess systems. *Computer Networks*, 2:95–114, 1978.

[Lam86] Leslie Lamport. The mutual exclusion problem: part i - a theory of interprocess communication. *J. ACM*, 33(2):313–326, 1986.

[Lam98] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998.

[Lap05] Jean-Claude Laprie. Resilience for the scalability of dependability. In *Proc. 4th Int. Symposium on Network Computing and Applications*, pages 5–6. IEEE CS, 2005.

[LFCL03] Deron Liang, Chen-Liang Fang, Chyouhwa Chen, and Fengyi Lin. Fault tolerant web service. In *Proc. 10th Asia-Pacific Software Engineering Conf.*, pages 310–319. IEEE CS, 2003.

[LGG+91] Barbara Liskov, Sanjay Ghemawat, Robert Gruber, Paul Johnson, and Liuba Shrira. Replication in the harp file system. *SIGOPS Oper. Syst. Rev.*, 25(5):226–238, 1991.

[Mad98] Sanjay Kumar Madria. Handling of mutual conflicts in distributed databases using timestamps. *The Computer Journal*, 41(6):376–385, 1998.

BIBLIOGRAPHY

[Mica] Microsoft. Distributed File System (DFS). http://www.microsoft.com/windowsserver2003/ technologies/storage/dfs/.

[Micb] Microsoft. Microsoft SQL Server,. http://www.microsoft.com/sql/.

[Micc] Microsoft. .NET Framework 3.0, http://msdn2.microsoft.com/ en-us/netframework/.

[Micd] Microsoft. Windows Communication Foundation (WCF), http://msdn2.microsoft.com/en-us/netframework/aa663324.aspx.

[MIM+05] Michael G. Merideth, Arun Iyengar, Thomas Mikalsen, Stefan Tai, Isabelle Rouvellou, and Priya Narasimhan. Thema: Byzantine-fault-tolerant middleware for web-service applications. In *Proc. 24th Symp. on Reliable Distributed Systems*, pages 131–142. IEEE CS, 2005.

[MMN98] Louise E. Moser, P. Michael Melliar-Smith, and Priya Narasimhan. Consistent object replication in the eternal system. *Theor. Pract. Object Syst.*, 4(2):81–92, 1998.

[MMVB00] Carlo Marchetti, Massimo Mecella, Antonino Virgillito, and Roberto Baldoni. An interoperable replication logic for corba systems. In *Proc. 2nd Int. Symp. on Distributed Objects and Applications (DOA)*, pages 7–16. IEEE CS, 2000.

[MMZ06] Louise E. Moser, P. Michael Melliar-Smith, and Wenbing Zhao. Making web services dependable. In *Proc. 1st Int. Conf. on Availability, Reliability, and Security*, pages 440–448. IEEE CS, 2006.

[Moc07] Jaka Mocnik (ed.). Software Prototype and Refined Design and Validation Report. Technical Report D3.1.2, DeDiSys Consortium (www.dedisys.org), 2007.

[MS88] Keith Marzullo and Frank Schmuck. Supplying high availability with a standard network file system. In *Proc. of the 8th Int. Conf. on Distributed Computing Systems*, pages 447–455. IEEE CS Press, 1988.

[MyS] MySQL AB. MySQL,. http://www.mysql.com.

[NMMS97] Priya Narasimhan, Louise E. Moser, and P. Michael Melliar-Smith. Exploiting the internet inter-orb protocol interface to provide corba with fault tolerance. In *COOTS'97: Proc. of the 3rd USENIX Conf. on Object-Oriented Technologies (COOTS)*, pages 6–6. USENIX Association, 1997.

[OAS] OASIS - Organization for the Advancement of Structured Information Standards. http://www.oasis-open.org.

[Obj07] ObjectWeb. What's middleware, 2007. http://middleware.objectweb.org.

BIBLIOGRAPHY

[OMG04] OMG - Object Management Group. Common Object Request Broker Architecture: Core Specification, v3.0.3, 2004.

[Ora] Oracle. Oracle Database,. http://www.oracle.com/database/.

[Pap03] Mike Papazoglou. Service-oriented computing: Concepts, characteristics and directions. In *Proc. 4th Int. Conf. on Web Information Systems Engineering*, pages 3–12. IEEE CS, 2003.

[PBM+00] Nuno Preguica, Carlos Baquero, Francisco Moura, Jose Legatheaux Martins, Rui Oliveira, Henrique Domingos, Jose Pereira, and Sergio Duarte. Mobile transaction management in mobisap. In *Current Issues in Databases and Information Systems*, volume 1884 of *LNCS*, pages 379–386. Springer, 2000.

[PCD91] David Powell, Marc Chéréque, and David Drackley. Fault-tolerance in delta-4. *SIGOPS Oper. Syst. Rev.*, 25(2):122–125, 1991.

[PHE+06] Srinath Perera, Chathura Herath, Jaliya Ekanayake, Eran Chinthaka, Ajith Ranabahu, Deepal Jayasinghe, Sanjiva Weerawarana, and Glen Daniels. Axis2, middleware for next generation web services. In *Proc. Int. Conf. on Web Services (ICWS'06)*, pages 833–840. IEEE CS, 2006.

[Pos] PostgreSQL Global Development Group. PostgreSQL,. http://www.postgresql.org.

[RBC+03] Jennifer Ren, David E. Bakken, Tod Courtney, Michel Cukier, David A. Karr, Paul Rubel, Chetan Sabnis, William H. Sanders, Richard E. Schantz, and Mouna Seri. Aqua: An adaptive architecture that provides dependable distributed objects. *IEEE Trans. on Computers*, 52(1):31–50, Jan. 2003.

[RDH05] Hans P. Reiser, Michael J. Danel, and Franz J. Hauck. A flexible replication framework for scalable and reliable .net services. In *Proc. of the IADIS Int. Conf. on Applied Computing*, volume 1, pages 161–169, 2005.

[Sch93] Fred B. Schneider. Replication management using the state-machine approach. In S.J. Mullender, editor, *Distributed Systems*, chapter 2, pages 17–26. ACM Press, Addison-Wesley, 2nd edition, 1993.

[Sch06] André 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.

[SCR+03] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck. Network File System (NFS) version 4 protocol. In *RFC3530*. RFC Editor, 2003.

BIBLIOGRAPHY

[SG03] Robert Smeikal and Karl M. Goeschka. Fault-tolerance in a distributed management system: a case study. In *Proc. 25th Int. Conf. on Software Engineering*, pages 478–483. IEEE, 2003.

[SG04] Robert Smeikal and Karl M. Goeschka. Trading constraint consistency for availability of replicated objects. In *Proc. of the 16th IASTED Int. Conf. on Parallel and Distributed Computing Systems (PDCS 2004)*, pages 232–237. ACTA Press, 2004.

[SKK+90] Mahadev Satyanarayanan, James J. Kistler, Puneet Kumar, Maria E. Okasaki, Ellen H. Siegel, and David C. Steere. Coda: A highly available file system for a distributed workstation environment. In *IEEE Transactions on Computers*, volume 39, pages 447–459. IEEE CS, 1990.

[Sme04] Robert Smeikal. *Trading Consistency for Availability in a Replicated System.* PhD thesis, Vienna University of Technology, 2004.

[SPPJ06] Jorge Salas, Francisco Perez-Sorrosal, Marta Patiño-Martínez, and Ricardo Jiménez-Peris. WS-Replication: a framework for highly available web services. In *Proc. 15th Int. Conf. on World Wide Web*, pages 357–366, New York, NY, USA, 2006. ACM Press.

[SS05] Yasushi Saito and Marc Shapiro. Optimistic replication. *ACM Comput. Surv.*, 37(1):42–81, 2005.

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

[Tho79] Robert H. Thomas. A majority consensus approach to concurrency control for multiple copy databases. *ACM Trans. Database Syst.*, 4(2), 1979.

[TIRL03] Ferda Tartanoglu, Valérie Issarny, Alexander B. Romanovsky, and Nicole Lévy. Dependability in the web services architecture. In *Architecting Dependable Systems*, LNCS 2677, pages 90–109. Springer, 2003.

[TTP+95] Douglas B. Terry, Marvin M. Theimer, Karin Petersen, Alan J. Demers, Mike J. Spreitzer, and Carl H. Hauser. Managing update conflicts in bayou, a weakly connected replicated storage system. In *Proc. 15th ACM Symp. on Operating Systems Principles*, pages 172–182. ACM, 1995.

[VB98] Alexey Vaysburd and Ken Birman. The maestro approach to building reliable interoperable distributed applications with multiple execution styles. *Theor. Pract. Object Syst.*, 4(2):71–80, 1998.

[VR03] Werner Vogels and Chris Re. Ws-membership - failure management in a web-services world. In *Proc. 12th Int. World Wide Web Conf. (Alternate Paper Tracks)*, 2003. http://www2003.org/cdrom/papers/alternate/P712/p712-vogels.html.

BIBLIOGRAPHY

[W3C] W3C - World Wide Web Consortium. http://www.w3.org.

[Weg07] Martin Weghofer. Implementierung und evaluierung von webservice replikationsmechanismen. Master's thesis, University of Applied Sciences Technikum Wien, 2007. Advisors: Karl M. Goeschka and Johannes Osrael.

[WKM04] Huaigu Wu, Bettina Kemme, and Vance Maverick. Eager Replication for Stateful J2EE Servers. In *Proc. OTM Federated Conf.*, volume 3291 of *LNCS*, pages 1376–1394. Springer, 2004.

[WPS+00] Matthias Wiesmann, Fernando Pedone, André Schiper, Bettina Kemme, and Gustavo Alonso. Understanding replication in databases and distributed systems. In *Proc. of 20th Int. Conf. on Distributed Computing Systems (ICDCS'2000)*, Taipei, Taiwan, R.O.C., 2000. IEEE CS.

[WSP+00] Matthias Wiesmann, André Schiper, Fernando Pedone, Bettina Kemme, and Gustavo Alonso. Database replication techniques: A three parameter classification. In *Proc. of the 19th IEEE Symposium on Reliable Distributed Systems (SRDS'00)*, page 206. IEEE CS, 2000.

[YS05] Xinfeng Ye and Yilin Shen. A middleware for replicated web services. In *Proc. 3rd Int. Conf. on Web Services*, pages 631–638. IEEE CS, 2005.

[YV02] Haifeng Yu and Amin Vahdat. Design and evaluation of a conit-based continuous consistency model for replicated services. *ACM Trans. Comput. Syst.*, 20(3):239–282, 2002.

[Zag07] Klemen Zagar (ed.). Software Prototype and Refined Design and Validation Report. Technical Report D3.4.2, DeDiSys Consortium (www.dedisys.org), 2007.

Curriculum Vitae

Johannes Osrael

Kurze Zeile 64
2212 Großengersdorf, Austria

Date of birth: July 25, 1980
Email: johannes@osrael.com

Education

2004 - 2007	Doctoral Studies in Computer Science, Vienna University of Technology
1999 - 2004	Diploma Studies (MSc.) in Electrical Engineering/Computer Technology, Vienna University of Technology
1995 - 1999	Technical High School "TGM Wien 20", Electronics/Biomedical Technologies

Work Experience

11/2004 -	Research Assistant, Vienna Univ. of Technology, Faculty of Informatics, Institute of Information Systems, Distributed Systems Group *Research focus:* dependable distributed systems, replication, middleware, system architectures *Project management:* leader of several international work packages *Organizational activities*: organized several international scientific workshops and conference tracks

10/2005 -	Lecturer at Univ. of Applied Sciences “FH Campus Wien”, several courses on database systems, supervision of MSc. students
03/2007 -	Lecturer at Univ. of Applied Sciences “Technikum Wien”, course on system integration, supervision of MSc. students

International Experience

11/2004 -	Held several talks (>10) at international conferences, participation in international research project DeDiSys (8 partners from 5 countries)
08/2003 - 12/2003	Exchange student at the University of Illinois at Urbana-Champaign, USA

Languages

German (native language), English (business fluent), French (basic knowledge)

Publications

[1] Johannes Osrael, Lorenz Froihofer, and Karl M. Goeschka. A replication model for trading data integrity against availability. In *Proc. of the 12th Pacific Rim Int. Symposium on Dependable Computing (PRDC'06)*, pages 377–378. IEEE CS, 2006.

[2] Johannes Osrael, Lorenz Froihofer, and Karl M. Goeschka. Availability/Consistency Balancing Replication Model. In *Workshop Proc. of the 21st Int. Parallel and Distributed Processing Symp.* IEEE CS, 2007.

[3] Johannes Osrael, Lorenz Froihofer, Matthias Gladt, and Karl M. Goeschka. Adaptive voting for balancing data integrity with availability. In *On the Move to Meaningful Internet Systems 2006: Confederated Int. Workshops Proc.*, volume 4278 of *LNCS*, pages 1510–1519. Springer, 2006.

[4] Johannes Osrael, Lorenz Froihofer, Norbert Chlaupek, and Karl M. Goeschka. Availability and performance of the adaptive voting replication protocol. In *Proc. of the 2nd Int. Conf. on Availability, Reliability and Security*, pages 53–60. IEEE CS, 2007.

[5] Johannes Osrael, Lorenz Froihofer, Karl M. Goeschka, Stefan Beyer, Pablo Galdámez, and Francesc Muñoz. A system architecture for enhanced availability of tightly coupled distributed systems. In *Proc. of the 1st Int. Conf. on Availability, Reliability and Security*, pages 400–407. IEEE CS, 2006.

[6] Johannes Osrael, Lorenz Froihofer, Georg Stoifl, Lucas Weigl, Klemen Zagar, Igor Habjan, and Karl M. Goeschka. Using replication to build highly available .NET applications. In *Workshop Proc. of the 17th Int. Conf. on Database and Expert Systems Applications*, pages 385–389. IEEE CS, 2006.

[7] Johannes Osrael, Lorenz Froihofer, and Karl M. Goeschka. What service replication middleware can learn from object replication middleware. In *Proc. of the 1st Workshop on Middleware for Service Oriented Computing in conjunction with the ACM/IFIP/USENIX Middleware Conf. 2006*, pages 18–23. ACM Press, 2006.

[8] Johannes Osrael, Lorenz Froihofer, and Karl M. Goeschka. *Software Engineering of Fault Tolerant Systems*, chapter Replication in Service-Oriented Systems. World Scientific Publishing, 2007.

[9] Johannes Osrael, Lorenz Froihofer, Martin Weghofer, and Karl M. Goeschka. Axis2-based replication middleware for Web services. In *Proc. of the Int. Conf. on Web Services*, pages 591–598. IEEE CS, 2007.

[10] Johannes Osrael, Lorenz Froihofer, and Karl M. Goeschka. Experiences from building service and object replication middleware. In *Workshop Proc. of the 6th Int. Symp. on Network Computing and Applications*, pages 305–310. IEEE CS, 2007.

[11] Johannes Osrael, Lorenz Froihofer, Hubert Kuenig, and Karl M. Goeschka. Scenarios for increasing availability by relaxing data integrity. In P. Cunningham and M. Cunningham, editors, *Innovation and the Knowledge Economy - Issues, Applications, Case Studies*, volume 2, pages 1396–1403. IOS Press, 2005.

[12] Johannes Osrael, Lorenz Froihofer, and Karl M. Goeschka. On the need for dependability research on service oriented systems. In *Fast Abstract Proc. of the 37th Int. Conference on Dependable Systems and Networks*. IEEE CS, 2007.

[13] Lorenz Froihofer, Johannes Osrael, and Karl M. Goeschka. Trading integrity for availability by means of explicit runtime constraints. In *Proc. of the 30th Int. Computer Software and Applications Conference*, pages 14–17. IEEE CS, 2006.

[14] Lorenz Froihofer, Johannes Osrael, and Karl M. Goeschka. Decoupling constraint validation from business activities to improve dependability in distributed object systems. In *Proc. of the 2nd Int. Conf. on Availability, Reliability and Security*, pages 443–450. IEEE CS, 2007.

[15] Lorenz Froihofer, Gerhard Glos, Johannes Osrael, and Karl M. Goeschka. Overview and evaluation of constraint validation approaches in java. In *Proc. of the 29th Int. Conf. on Software Engineering*, pages 313–322. IEEE CS, 2007.

[16] Stefan Beyer, Mari-Carmen Bañuls, Pablo Galdámez, Johannes Osrael, and Francesc Muñoz. Increasing availability in a replicated partitionable distributed object system. In *Proc. of the 4th Int. Symp. on Parallel and Distr. Processing and Appl. (ISPA'06)*, volume 4330 of *LNCS*, pages 682–695. Springer, 2006.

[17] Lorenz Froihofer, Johannes Osrael, and Karl M. Goeschka. Middleware application interactions to support adaptive dependability. In *Proc. of the 1st Workshop on Middleware Application Interaction in conjunction with EuroSys'07*, pages 31–36. ACM Press, 2007.

[18] Lorenz Froihofer, Karl M. Goeschka, and Johannes Osrael. Middleware support for adaptive dependability. In *Proc. of the ACM/IFIP/USENIX 8th Int. Middleware Conference*, LNCS. Springer, 2007.

[19] Lorenz Froihofer, Markus Baumgartner, Johannes Osrael, and Karl M. Goeschka. Data partitioning through integrity constraints. In *Fast Abstract Proc. of the 37th Int. Conference on Dependable Systems and Networks*. IEEE CS, 2007.