How to Implement Multi-Master Replication in Polyhedra

Using Full Replication and Eventual Consistency

Sebastian Holmgren
How to Implement Multi-Master Replication in Polyhedra
Using Full Replication and Eventual Consistency

Submitted by Sebastian Holmgren to the University of Skövde, as a dissertation for the degree of Master of Science (M.Sc.) by examination and dissertation at the School of Humanities and Informatics.

November 18, 2006

I certify that all material in this dissertation which is not my own work has been clearly identified and that no material is included for which a degree has previously been conferred on me.

Signed: ________________________________________________

Supervisor: Marcus Brohede
Examiner: Sten F. Andler
How to Implement Multi-Master Replication in Polyhedra
Using Full Replication and Eventual Consistency

Sebastian Holmgren

Abstract

A distributed, real-time database could be used to implement a shared whiteboard architecture used for communication between mobile nodes, in an ad-hoc network. This kind of application implies specific requirements on how the database handles replication and consistency between replicas (global consistency). Since mobile nodes are likely to disconnect from the network and connect again at unpredictable times, and since a node may be disconnected an arbitrary amount of time, this needs to be treated as normal operation, and not as failures.

The replication scheme used in the DeeDS architecture, and the PRiDe replication protocol are both suitable for a shared whiteboard architecture as described above. Since the mobile nodes are likely to be some kind of hand-held device (e.g., used by rescue personnel to exchange information), the database system should be suitable for use in embedded systems. The Polyhedra Real-Time Relational Database (RTRDB) and the TimesTen database are two such systems. A problem is that neither of these two database systems have a replication scheme suitable for use in the previously described type of architecture.

This dissertation presents two design proposals for how to extend the Polyhedra RTRDB with support for multi-master replication of data using full replication and eventual consistency. One design proposal is based on the DeeDS architecture and the other is based on the PRiDe replication protocol. The proposal based on DeeDS puts a number of requirements on the underlying database and is not easy to port to another DBMS since it makes use of Polyhedra specific API’s. The proposal based on PRiDe on the other hand requires no instrumentation of the underlying database and is thus easier to port to other database systems.

Keywords: Real-Time Databases, Multi-Master Replication, Full Replication, Eventual Consistency, DeeDS, PRiDe, Polyhedra, TimesTen
Acknowledgements

I wish to express my gratitude to the drts-research group at the University of Skövde. In particular to my supervisor Marcus Brohede for all his help and guidance with my dissertation. Further, Sanny Gustavsson and Gunnar Mathiasen have shared their view of my dissertation. Sanny has also given me valuable insight in the inner workings of eventual consistency, the DeeDS architecture and the PRiDe replication protocol.

I would also like to thank my examiner, professor Sten F. Andler for his help during the dissertation, professor Sang H. Son for sharing his thoughts about the ideas behind the dissertation, Dr. Bengt Eftring for reviewing the dissertation, and Dr. Ronnie Johansson for opposing against the dissertation. Further, I want to express my gratitude towards Polyhedra Ltd and especially Nigel Day for all the information about Polyhedra and for sharing their thoughts about how multi-master data replication could be implemented in the Polyhedra RTRDB. Last, but not least I would like to thank my wife Karin, for all her support when I wrote this dissertation.
## Contents

1 Introduction ............................................. 1

2 Background .................................................. 2
   2.1 Real-Time Database Systems ............................. 2
      2.1.1 Consistency vs. Predictability ...................... 2
   2.2 Replication & Consistency ............................... 2
      2.2.1 Replication Schemes ................................. 3
      2.2.2 Consistency Guarantees .............................. 4
   2.3 The DeeDS Architecture ................................. 5
   2.4 The PRiDe Replication Protocol ......................... 5
   2.5 The TimesTen In-Memory Database ....................... 6
   2.6 The Polyhedra RTRDB .................................. 7
   2.7 Example Scenario ..................................... 8
      2.7.1 Problems with Leader-follower Replication .......... 8

3 Problem Statement .......................................... 10
   3.1 Aim .................................................... 10
   3.2 Limitations ............................................ 10
   3.3 Objectives ............................................. 11

4 Research Approach .......................................... 13

5 Investigations ............................................. 14
   5.1 Replication in DeeDS ................................. 14
      5.1.1 The Replication Module ............................ 14
      5.1.2 Local Conflict Detection & Resolution ............. 15
      5.1.3 Key Design Choices in the DeeDS Architecture ..... 16
   5.2 The PRiDe Replication Protocol ......................... 16
      5.2.1 Stable and Optimistic Versions ..................... 17
      5.2.2 Generations & Conflict Sets ....................... 17
      5.2.3 Semantic Conflict Detection & Resolution .......... 18
      5.2.4 Steps in PRiDe Replication ......................... 18
   5.3 Replication in the TimesTen In-Memory Database ......... 18
      5.3.1 Replication Schemes ............................... 19
      5.3.2 Consistency Guarantees ............................ 19
      5.3.3 Conflict Detection & Resolution .................. 20
   5.4 Replication in the Polyhedra RTRDB ..................... 20

6 Comparisons ............................................... 22
   6.1 Comparison between DeeDS and PRiDe .................... 22
   6.2 Comparison of DeeDS, DeeDS NG, TimesTen and Polyhedra ... 23
      6.2.1 Conflict Types .................................... 24

7 Extension Designs ........................................... 25
   7.1 Based on DeeDS ........................................ 25
      7.1.1 Prerequisites for Multi-Master Replication ........ 25
      7.1.2 High Level Design ................................ 26
      7.1.3 Requirements on Polyhedra ......................... 27
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1.4 Architectural Design</td>
<td>27</td>
</tr>
<tr>
<td>7.1.5 Replicating an Update Operation</td>
<td>27</td>
</tr>
<tr>
<td>7.2 Based on PRiDe</td>
<td>29</td>
</tr>
<tr>
<td>7.2.1 Object Level Interface</td>
<td>29</td>
</tr>
<tr>
<td>7.2.2 Mapping from Object Level to Relational Level</td>
<td>30</td>
</tr>
<tr>
<td>7.2.3 Moving Requirements</td>
<td>30</td>
</tr>
<tr>
<td>8 Discussion</td>
<td>31</td>
</tr>
<tr>
<td>8.1 Portability</td>
<td>31</td>
</tr>
<tr>
<td>8.2 Advantages &amp; Disadvantages of the Design Proposals</td>
<td>31</td>
</tr>
<tr>
<td>8.3 Application - MMR/EC Interface</td>
<td>31</td>
</tr>
<tr>
<td>8.4 Propagation Strategies</td>
<td>32</td>
</tr>
<tr>
<td>9 Conclusions</td>
<td>33</td>
</tr>
<tr>
<td>9.1 Summary</td>
<td>33</td>
</tr>
<tr>
<td>9.2 Contributions</td>
<td>34</td>
</tr>
<tr>
<td>9.3 Related Work</td>
<td>34</td>
</tr>
<tr>
<td>9.4 Future Work</td>
<td>34</td>
</tr>
<tr>
<td>References</td>
<td>36</td>
</tr>
</tbody>
</table>
List of Figures

1 The DeeDS architecture (adapted from Andler, Hansson, Mellin, Eriksson & Eftring (1998)) .............................................. 5
2 TimesTen components (adapted from Oracle (2006a)) .......... 6
3 The structure of the Polyhedra RTRDB (adapted from ENEA Polyhedra (2005b)) .................................................. 7
4 Search and rescue scenario .................................................. 8
5 Replication in DeeDS (adapted from Matheis & Müssig (2003)) 14
6 Conflict detection in DeeDS .................................................. 16
7 TimesTen replication (adapted from Oracle (2006b)) ................. 19
8 Fault-tolerant configuration (adapted from ENEA Polyhedra (2005b)) 21
9 Proposed design for MMR/EC support in the Polyhedra RTRDB, based on DeeDS ......................................................... 28
10 Replication of an update operation ....................................... 28
11 Proposed design for MMR/EC support in the Polyhedra RTRDB, based on PRiDe ......................................................... 29

List of Tables

1 Comparison of replication in DeeDS and the PRiDe replication protocol 22
2 Comparison of replication schemes ........................................ 23
3 Conflict types in TimesTen and the DeeDS ................................ 24
4 Advantages & disadvantages for the two design proposals. ............ 31
1 Introduction

Real-time databases are used by real-time applications since ad-hoc management of data is not desirable and conventional databases have unpredictable response times. To improve fault-tolerance and availability decentralised configurations and replication are often used. When data are replicated onto different nodes, the consistency between the replicas becomes an issue. The concept of eventual consistency allows temporary inconsistencies between replicas; eventually, given transaction quiescence, the replicas will converge into a consistent state. In this way, a local application may still access the local database replica without waiting for other nodes, thereby increasing predictability, speed and node autonomy while still guaranteeing global eventual consistency. The price for this is that applications must tolerate temporary global inconsistencies in the database.

A distributed, real-time database could be used to implement a shared whiteboard architecture (Brohede & Andler 2002) used for communication between mobile nodes in an ad-hoc network (Tatomir & Rothkrantz 2005). This kind of application implies specific requirements on how the database handles replication and consistency between replicas (global consistency). Since mobile nodes are likely to disconnect from the network and connect again at unpredictable times, and since a node may be disconnected an arbitrary amount of time, this needs to be treated as normal operation of the system, and not as failures (Serrano-Alvarado, Roncancio & Adiba 2004).

The DeeDS architecture and the PRiDe replication protocol are both targeted towards hard real-time systems where predictability is critical. For this reason, they both support local commits while guaranteeing eventual global consistency, and local conflict detection and resolution. An observation is that the approach taken in these systems for improving predictability (e.g., multi-master replication, full replication, and eventual consistency) is also suitable for a shared whiteboard architecture used for communication as described above.

Since the mobile nodes are likely to be some kind of hand-held device (e.g., used by rescue personnel to exchange information), the database system should be suitable for use in embedded systems. The Polyhedra Real-Time Relational Database (RTRDB) is one such system. The TimesTen database system is also suitable for use in embedded system. Further, the replication scheme in TimesTen shares a number of features with the replication scheme used in the DeeDS architecture and the PRiDe replication protocol. However, some important features, such as detecting read-write conflicts, are lacking from the TimesTen replication scheme (discussed in Section 6.2) and therefore the TimesTen database is not sufficient for use instead of the DeeDS architecture. Thus the aim of this dissertation is to investigate how multi-master replication using full replication and eventual consistency can be integrated in the Polyhedra RTRDB.

Section 2 presents the central concept of this dissertation along with important terminology used throughout the dissertation. Section 3 presents and motivates the aim for the dissertation along with a number of objectives which need to be met in order to reach the aim. Section 4 presents the research methods chosen for each objective. Section 5 presents a more in-depth investigation of the DeeDS, PRiDe, TimesTen and Polyhedra systems. These systems are then compared in section 6. Section 7 presents two design proposals (MMR/EC based on the DeeDS architecture and MMR/EC based on the PRiDe replication protocol) for how to extend the Polyhedra RTRDB with support for multi-master replication. Section 8 discusses the work presented in the dissertation along with some of the results presented earlier. Finally section 9 concludes the report and presents the contributions made in this dissertation along with related and future work.
2 Background

This section provides an overview of real-time database systems and why they are used. Further, replication, and consistency guarantees are described. Three real-time database systems, the DeeDS distributed real-time database architecture, the TimesTen database and the Polyhedra real-time relational database (RTRDB), along with the PRiDe replication protocol are also described.

2.1 Real-Time Database Systems

Real-time database systems are used because ad-hoc management of data are not desirable; moreover, conventional (non real-time) database systems do not meet the requirements of timeliness and predictability needed by real-time systems (Gustavsson & Andler 2005, Ramamritham 1993, Ramamritham, Son & Dippippo 2004, Stankovic, Son & Hansson 1999). Note that timeliness and predictability has nothing to do with speed. Instead timeliness means that all deadlines are met, that is a timely system does not miss any deadlines. Further, predictability means that response time for the worst case can be predicted.

In a real-time database system, transactions are associated with deadlines, and data may be valid for specific time intervals (Bestavros, Son & Lin 1997, Ramamritham 1993, Stankovic et al. 1999). We associate transactions with deadlines because since real-time systems may be depending on timely access to data in a database. Transaction execution within the database must also be timely. As in a real-time system, deadlines might be soft, firm or hard (Bernat, Burns & Llamosi 2001).

2.1.1 Consistency vs. Predictability

Conventional databases follow the ACID (atomicity, consistency, isolation, durability) properties (Elmasri & Navathe 2000). In a real-time database system, the ACID properties may be traded off. Especially consistency is often traded off for better predictability and timeliness. If a distributed and replicated real-time database were to enforce immediate consistency, this would lead to unpredictability, unless a real-time network is used, since it would have to access data over the network and a global commit protocol would have to be used. Instead, some real-time databases guarantee eventual consistency. Even if a real-time network is available, consistency may be sacrificed in order to speed up the time it takes to commit (i.e., global commit over the network may still be slow, even if it is predictable).

2.2 Replication & Consistency

Real-time systems are often used in safety critical applications (e.g., fighter planes, air traffic control, nuclear power plants). This means that the system needs to be safe (nothing bad happens), reliable (the system does what it is supposed to do), available (the system can be accessed and used by an application) and fault-tolerant. To achieve availability and fault-tolerance, distributed configurations together with replication is often employed (Birrell, Levin, Schroeder & Needham 1982, Gray, Helland, O’Neil & Shasha 1996, Gustavsson & Andler 2002, Joseph & Birman 1986, Mathiason 2002, Mathiason & Andler 2003).

Saito & Shapiro (2005, pp. 1-2) state:

Data replication consists of maintaining multiple copies of data, called replicas, on separate computers. It is an important enabling technology for distributed
services. Replication improves availability by allowing access to the data even when some of the replicas are unavailable.

In this dissertation, the term *replica* refers to a replica of an entire database, assuming full replication. An object in such a database will be referred to as a *replicated object*.

The reason for replicating data is that it improves performance by avoiding remote network access, and the availability by allowing access to data even though some of the replicas are unavailable (Saito & Shapiro 2005, Gray et al. 1996). Further, according to Ramamritham et al. (2004), “Many real-time database applications are inherently distributed in nature”. The availability, performance and reliability of such applications can be significantly enhanced by replicating data.

When data are replicated in order to achieve availability and fault-tolerance, *full replication* is often used. In a fully replicated database system, all data items are stored on all nodes in the system and thus, full availability and better fault-tolerance is achieved. However, this leads to poor scalability since all updates must be sent to all nodes even if the affected data are never used at some nodes. Moreover, full replication is costly in storage requirements. *Virtual full replication* (Andler, Hansson, Eriksson, Mellin, Berndtsson & Eftring 1996, Andler et al. 1998) can be used to reduce this problem in order to improve scalability without changing the application’s view of a fully replicated database system (Mathiason & Andler 2003, Mathiason, Andler & Jagszent 2005).

With virtual full replication, data are replicated only to nodes where they are used (Mathiason & Andler 2003). This reduces the amount of data stored on each node and the need to replicate updates to all nodes, thus improving scalability while still offering better availability and fault-tolerance. If a database is virtually fully replicated, the term replica will refer to the parts of the replicated database that are replicated on one of the nodes.

### 2.2.1 Replication Schemes

There are many different approaches or schemes for how to update data in a replicated system. Two such schemes are (i) the leader-follower replication scheme, and (ii) the multi-master replication scheme.

If the *leader-follower* replication scheme (Barret, Hilborne, Bond, Seaton, Verissimo, Rodrigues & Speirs 1990) is used, one replica is elected to be the leader. This is also referred to as *single-master* replication (Saito & Shapiro 2005). In this dissertation, the term leader-follower replication will be used. All clients then issue their *update operations* (e.g., insert, update, delete) to the leader replica. The leader will then propagate the update operations to the follower replicas. Read operations can be issued to any replica. If a global commit protocol is used, all replicas will be “fresh” when read by an application. Otherwise, there is a risk that an application reads “stale” data (i.e., an operation has updated data on the leader but it has not yet been propagated to the follower replica(s)) from the follower. If the leader fails, the follower replicas elect a new leader.

In this dissertation, the term *update operation* (e.g., insert, update or delete) will be referred to as an *update*. Thus, an update is an operation that affect the state of the database. Read operations will be referred to as reads, this type of operation does not affect the state of the database. Finally the term *operations* will be used to denote both updates and reads.

The *multi-master* replication scheme (Saito & Shapiro 2005) differs from the leader-follower scheme in that clients can issue their operations to more than one replica, thereby avoiding the single-point-of-failure of the leader replica. When accepting operations on any replica in the system, it is possible to let the replica that receives the operation commit the
change locally. This allows the client continue working based on that operation, and then propagate the change to the other replicas in the background.

### 2.2.2 Consistency Guarantees

When data are replicated on separate nodes in a distributed network, the consistency of the replicated data becomes an issue (Fekete, Gupta, Luchangco, Lynch & Shvartsman 1996). Keeping a distributed database consistent requires keeping replicas sufficiently similar to each other despite operations being submitted independently at different sites (Saito & Shapiro 2005). In this dissertation, two different kinds of consistency are described: local consistency and global consistency. Local consistency refers to the internal consistency (Thomas 1979) of one replica. According to Thomas (1979, p. 181), internal consistency concerns “the preservation of invariant relations that exist among items in a items within a database”. Global consistency on the other hand refers to mutual consistency (Thomas 1979) between replicas in a replicated database system. Mutual consistency is described by Thomas (1979, p. 181) “all replicas converge to the same state and would be identical should update activity cease”. In this dissertation, the terms local consistency and global consistency will be used.

The global consistency guarantee defines how much replica divergence a client application may observe at a given moment. Different systems offer different consistency guarantees. Three different levels of consistency guarantees are common (Saito & Shapiro 2005):

- Single-copy consistency, also referred to as immediate (global) consistency. In this dissertation, the term immediate (global) consistency will be used. In TimesTen, the return twosafe variant of synchronous replication is equivalent to immediate consistency.

- Bounded divergence, also referred to as bounded replication.

- Eventual consistency. In TimesTen, the asynchronous replication resembles eventual consistency in that an operation is directly committed and later propagated to other replicas.

Immediate global consistency guarantees that an application that uses the replicated data never sees any inconsistencies between the different replicas. Instead, it appears to the application that it is using one single, highly available source of data (Saito & Shapiro 2005).

Bounded divergence is a weaker guarantee in the sense that it lets applications see inconsistencies in the replicated data. The replicas are allowed to temporarily diverge, but replicas are guaranteed to converge to a consistent state within a specified amount of time (Yu & Vahdat 2001).

Eventual consistency is an even weaker guarantee than bounded divergence. Also in this case, replicas are allowed to diverge. Eventual consistency guarantees that, given transaction quiescence, the replicas will eventually converge. Since replicas might be temporarily inconsistent, the application must tolerate this (Andler et al. 1996, Saito & Shapiro 2005).

In a fully replicated system that uses eventual consistency, a node can commit a transaction locally, without informing the other replicas, thereby avoiding global deadlocks. This implies the replicas will become temporarily inconsistent. That is, there is a trade-off between predictability (or speed cf. Section 2.1.1) and consistency. According to (Fox & Brewer 1999, Pedone 2001, Yu & Vahdat 2002) any distributed system must make such a trade-off.
2.3 The DeeDS Architecture

The DeeDS (Distributed, active, real-time Database System) research prototype (Andler, Hansson, Eriksson & Mellin 1994, Andler, Berndtsson, Eftring, Eriksson, Hansson & Mellin 1995, Andler et al. 1996, Andler et al. 1998) is an active main memory resident real-time database system that supports hard real-time requirements. It supports multi-master replication, currently using full replication, and support for virtual full replication (Section 2.2) is currently being researched (Mathiason & Andler 2003, Mathiason et al. 2005). Figure 1 depicts the DeeDS architecture.

![DeeDS Architecture Diagram]

Figure 1: The DeeDS architecture (adapted from Andler et al. (1998))

Besides support for eventual consistency, the DeeDS architecture also supports bounded replication. When eventual consistency is used, updates are propagated and integrated as soon as possible by using ASAP versions of the propagator and integrator modules. Bounded replication on the other uses bounded versions of the propagator and integrator which are predictable. Bounded propagation also requires a real-time network. In this dissertation only eventual consistency (ASAP replication) is considered.

DeeDS uses a combination of full replication and eventual consistency, which is referred to in the DeeDS project as lazy replication. Updates are accepted at any replica (multi-master replication), and thus the number of masters is equal to the number of nodes. Lazy replication, together with the fact that DeeDS resides in main memory removes some of the dominating sources of unpredictability, such as: disk I/O, network I/O (if no real-time network is available) and distributed commit protocols (e.g., two-phase commit (Gray, Flynn, Jones, Lagally, Opderbeck, Popek, Randell, Saltzer & Wiehle 1978, Gray & Lamport 2006)). This allows DeeDS to support hard real-time requirements locally, while gaining fault-tolerance and availability due to distribution and replication.

2.4 The PRiDe Replication Protocol

PRiDe (Protocol for Replication in DeeDS) is an optimistic replication protocol, intended for use in the DeeDS architecture. An initial version of the protocol, called the continuous convergence protocol is presented in (Gustavsson & Andler 2005). PRiDe is a further development of this protocol and will be presented in a future thesis (Gustavsson n.d.).
PRiDe uses a notion of *continuous convergence* (Gustavsson & Andler 2005) to maintain three important database properties: (i) local consistency, (ii) local predictability and (iii) eventual global consistency. This resembles the ideas behind replication in the DeeDS architecture. However, in the DeeDS architecture, operations are directly performed against the local replica of the database, transactions are monitored within the database and changes are logged and propagated to other nodes holding a replica of the database. Operations made to the local replica may be in conflict with remote operations on other nodes. These conflicts are then detected and resolved in the integration process. In PRiDe, on the other hand, there is a notion of *stable* and *optimistic* versions of replicas. Operations issued by applications are considered *tentative* and are kept within the data structures of PRiDe. When all conflicts have been detected and resolved for a specific operation it is considered *stable* and is integrated in the local database replica. It is now guaranteed that this update will not need to be corrected due to a conflict. The data which has been integrated on the database replica makes up the *stable state* of the replica. The stable state together with the tentative operations makes up the *optimistic state* of the replica.

PRiDe allows for more elaborate conflict detection and resolution than the DeeDS architecture by using *semantic information* as well as syntactic.

DeeDS NG (DeeDS N*ext G*eneration) (Andler, Brohede, Gustavsson & Mathiason 2007) refers to the DeeDS architecture complemented with the PRiDe protocol and virtual full replication.

### 2.5 The TimesTen In-Memory Database

TimesTen is an in-memory RDBMS. It is available as either a library that can be linked by applications and as a client server option (The TimesTen Team 2002, The TimesTen Team 1999). Although the database resides in main memory, it is persistent and recoverable. This is achieved by check pointing and logging to disk. One issue which is targeted by the TimesTen database is that of availability. By replicating the database, it is possible to access data even in the event of software or hardware failures. Further, on-line maintenance of replicas is supported.

![TimesTen components](adapted-from-Oracle(2006a))

The TimesTen database supports different kinds of replication configurations (active-passive and active-active), and consistency guarantees can be configured to be either asyn-
2.6 The Polyhedra RTRDB

The Polyhedra real-time relational database (RTRDB) is a main memory database designed for applications with rapidly changing data (ENEA Polyhedra 2005a).

Like DeeDS, the Polyhedra RTRDB resides in main memory. In contrast to DeeDS, the main reason for this is not predictability, but speed. Polyhedra gives no hard real-time guarantees, instead soft real-time guarantees are provided.

Figure 3: The structure of the Polyhedra RTRDB (adapted from ENEA Polyhedra (2005b))

Polyhedra is a single-master system (Section 2.2.1) and uses a leader-follower replication scheme with immediate global consistency ¹ (ENEA Polyhedra 2005b). Updates to the database are made at the leader (master replica), and are then propagated to the passive follower replica(s).

Figure 3 depicts the structure of the Polyhedra RTRDB. It consists of several modules. The schema contains the definitions of the current database, the object store contains the run-time data.

¹After a transaction has completed on the master replica, it sends journal records to the standby replica(s), and can be instructed to wait for the standby replica(s) to commit them before confirming to the calling application that the transaction is complete (Polyhedra 2005b)
2.7 Example Scenario

To understand why it is in some cases beneficial to support multi-master replication (Section 2.2.1), full replication and eventual consistency in a distributed real-time database, consider the following scenario.

Imagine a search and rescue scenario where a number of units (e.g., rescue personnel, vehicles etc.) are working together to perform some task (e.g., looking for survivors after a flood, avalanche, wildfire etc.). Figure 4 depicts a possible snapshot of the scenario. Besides radio communication, the units are equipped with some kind of hand-held device (e.g., a PDA) which is used to enter observations such as position of survivors or dangerous objects. A rescue worker could then get visual information about observations made by other rescuers. Communication between the devices is wireless. Further the network is dynamic in the sense that nodes may continuously join and leave the network. Since nodes may frequently be disconnected from the other nodes, this needs to be treated as normal operation and not as a failure (Serrano-Alvarado et al. 2004). This means that since nodes may not be able to reach each other from time to time, nodes need to be able to work autonomously. This requires access to data, and being able to commit locally.

To enable the different participants to communicate via the hand-held devices, Tatamir & Rothkrantz (2005) suggests that a shared whiteboard architecture can be used. In a whiteboard architecture, participants write data to a common place, the whiteboard. All participants can read and update the data. One of the advantages of such an architecture is that a participant that writes data does not need to know about how many other participants there are, or where they reside. They argue that such a whiteboard architecture is difficult to implement, and that this is mainly because the nodes in the network might not be able to reach each other from time to time since the network is mobile ad-hoc and wireless, i.e., a MANET (Corson & Macker 1999).

Brohede & Andler (2002) argues that a distributed, real-time database could be used to implement a shared whiteboard architecture useful for complex sharing applications such as distributed real-time simulations. If the database supports multi-master replication with (virtual) full replication and eventual consistency, this would be a suitable for use as a whiteboard architecture for communication in this type of environment. Another advantage of replicating the database, is that its availability and fault-tolerance is also increased.

![Figure 4: Search and rescue scenario](image)

2.7.1 Problems with Leader-follower Replication

If the leader-follower replication scheme was used, together with immediate global consistency, in the distributed database, the following two situations can cause problems: 

(i)
if the leader is unable to connect to one or more of the follower replicas, for instance the unavailable follower replica is in a ravine (i.e., radio shadow), and (ii) if a unit that wants to issue an update is unable to connect to the leader (e.g., they are too far away from each other or any intermediate communication node).

In the first case, if a unit issues an update (e.g., about its position), the leader would try to commit the update, and to replicate the change on all the followers. Suppose that one replica is currently unavailable. This could lead to that the leader would then have to wait for the follower to become available again (since all followers should be updated due to the immediate global consistency constraint), and reply before new updates can be taken care of, perhaps indefinitely.

In the second case, if a unit wants to issue an update but is unable to connect to the leader (directly or by using relays), it would not be possible for the unit to do so, even if it could connect to one or more of the follower replicas.
Section 2.7 describes a search and rescue scenario in which rescuers and various vehicles are cooperating. To enable the different participants to communicate, Tatamir & Rothkrantz (2005) suggests that a shared whiteboard architecture can be used. They argue that such a whiteboard architecture is difficult to implement, and that this is mainly because the nodes in the network might not be able to reach each other from time to time since the network is mobile, ad-hoc and wireless (i.e., a MANET).

Brohede & Andler (2002) argues that a distributed, real-time database could be used to implement a shared whiteboard architecture useful for complex sharing applications such as distributed real-time simulations. In this dissertation, we argue that with some modifications, such an architecture will also be useful for communication between mobile participants over an unreliable network.

Both the replication scheme used in the DeeDS architecture, and the PRiDe replication protocol provides a high degree of node autonomy by letting applications perform operations against their local replica of the distributed database without communicating with replicas on other nodes. This maps well against the characteristics for the scenario described above and in section 2.7. This leads to the conclusion that the DeeDS architecture would be suitable for use as a whiteboard architecture in such a scenario. However, as rescue workers are likely to be equipped with hand-held devices such as PDA’s, the whiteboard architecture should be suitable for use in embedded systems. According to Nyström, Tesanovic, Nolin, Norström & Hansson (2004) the DeeDS architecture is targeted against large scale real-time applications and thus it is unsuitable for use in embedded systems. Instead a database system suitable for embedded systems such as the Polyhedra RTRDB or the TimesTen database could be used if their replication schemes are suitable.

Currently, Polyhedra utilises the leader-follower replication scheme, and fault-tolerant pair configurations (see Section 5.4 and Figure 8) in order to tolerate node failures (ENEA Polyhedra 2005b). In Section 2.7.1 a number of problems that could occur when leader-follower replication is used are presented. The problems are: (i) that an unavailable replica would block the system from making any progress, and (ii) that a unit that is not able to connect to the leader could not issue any updates. Thus, this is not a suitable replication scheme for use in a whiteboard architecture. However, Polyhedra could be extended with support for the kind of replication scheme used in the DeeDS architecture or in PRiDe.

The TimesTen database system supports updates at more than one replica, and can be configured to use asynchronous replication which resembles the eventual consistency guarantee offered by the DeeDS architecture and PRiDe. Thus TimesTen is a suitable candidate and its replication scheme will be be evaluated further in Section 5.3.

3.1 Aim

The aim of this M.Sc. dissertation is to:

*Investigate how multi-master replication using full replication and eventual consistency could be implemented in a main-memory RDBMS such as Polyhedra.*

3.2 Limitations

As stated in Section 2.3 the DeeDS architecture supports two classes of replication, namely eventual consistency using best effort mechanisms and bounded delay replication; this dissertation, does not consider the bounded alternative. Since the Polyhedra RTRDB does
not support hard real-time requirements, providing bounded delay replication is not the top priority. However, future work may well extend the multi-master replication extension to support bounded replication, this is discussed in Section 9.4.

The focus of this dissertation is on how the features of the DeeDS architecture could be integrated in another database system, i.e., the Polyhedra RTRDB. Lower levels, such as communication protocols (e.g., for propagation) are only discussed briefly.

### 3.3 Objectives

To design such an extension, a number of objectives need to be met. These include:

1. **Transfer the Replication Scheme from DeeDS to Polyhedra**
   
   (a) **Investigate how Replication is Implemented in DeeDS**
   
   The research ideas behind the DeeDS architecture are used as guidelines. Hence, an investigation of how multi-master replication is designed and implemented in the DeeDS system is necessary. To fulfil this objective, all design decisions related to replication need to be identified and understood.

   (b) **Analyse the Polyhedra RTRDB**
   
   This objective consists of two parts: (i) Investigate how replication is implemented in the Polyhedra RTRDB and identify what is missing in order to support multi-master replication, and (ii) analyse the prerequisites and requirements for extending the Polyhedra RTRDB with multi-master replication and eventual consistency. For each of the design decisions identified in objective 1a, it should be investigated how to map it onto the Polyhedra RTRDB. For doing this, knowledge about how the Polyhedra RTRDB works (e.g., the transaction model, how events in the database can be caught, and how the database can be updated) needs to be established.

   (c) **Design the Extension**
   
   Using the results of objectives 1a and 1b a design based on the replication scheme used in the DeeDS architecture for the Polyhedra RTRDB should be created.

2. **Transfer the Replication Scheme from PRiDe to Polyhedra**

   (a) **Investigate how Replication is Implemented in PRiDe**

   An alternative design based on the PRiDe replication protocol is proposed. The PRiDe replication protocol is an alternative to the replication scheme used in the DeeDS architecture and is likely to replace it in future versions of DeeDS. This objective should investigate how replication works using PRiDe and identify important differences compared to the replication scheme currently used in the DeeDS architecture.

   (b) **Design the Extension**

   Using the results of objectives 2a and 1b a design based on the PRiDe replication protocol for the Polyhedra RTRDB should be created.

3. **Evaluate the Replication Scheme used in TimesTen**

   Since the TimesTen database can be configured to support a form of replication which accepts updates on multiple replicas, and to use an asynchronous consistency guarantee (Section 2.2.2), it should be evaluated if the replication scheme offered by
the TimesTen database could be used instead of the DeeDS replication scheme. If this is not the case, it should be identified what the TimesTen database is missing.
4 Research Approach

Section 3 stated the aim for this dissertation, and a number of objectives which should be met in order to reach the aim were formulated. In this section, the research methods chosen for each objective are described, and the sections where the objective is realised is presented.

When conducting a project in the area of computer science, there are several different approaches that we can choose from. Some of these are literature analysis, simulation, experiments, analysis or implementation (Berndtsson, Hansson, Olsson & Lundell 2002).

For this dissertation, a combination of the literature analysis method and the analysis method has been chosen. The literature analysis is carried out in order to gather background information about the key concepts for this dissertation, such as replication models and consistency guarantees.

A number of the objectives presented in Section 3.3 require detailed knowledge of how different mechanisms, such as replication or transaction handling, work in the different systems. Section 5 aims to provide detailed knowledge about such things; this is done by using the literature analysis method. Section 5.1 gives a detailed presentation of how replication is implemented in the DeeDS architecture (objective 1a), Section 5.2 provides similar information for the PRiDe replication protocol (objective 2a). Section 5.3 presents the replication scheme used in the TimesTen database (objective 3). The analysis method is then used to determine if the TimesTen replication scheme can be used instead of the DeeDS replication scheme (objective 3). Section 5.4 presents the replication scheme used in the Polyhedra RTRDB; information about how support for multi-master replication using eventual consistency can be included in Polyhedra is presented in Section 7.1 (objective 1b).

After the required information has been extracted the analysis method is used to create two design proposals for extension of the Polyhedra RTRDB: Section 7.1 presents a design for how to implement multi-master replication using eventual consistency based on the DeeDS architecture (objective 1c) and Section 7.2 presents the second design proposal which is based on PRiDe (objective 2b).
5 Investigations

In this section, the different systems are investigated.

5.1 Replication in DeeDS

As stated in Section 2.3, the DeeDS architecture supports multi-master replication. The replication scheme adopted in DeeDS (referred to as lazy replication) is used in order to make real-time database access predictable (Andler et al. 1995). That is, by replicating all data onto every node and accepting updates on every node, an application will never have to access data over the network. Instead it can always access and change data locally. Further, since eventual consistency is employed, updates can be made on different nodes simultaneously without any need to lock the entire database, only the local replica. When an update has committed locally, it is propagated to the other nodes. Thus, an application can always apply any changes necessary locally, without waiting for the remote replicas. This implies that the database may become temporarily globally inconsistent, and that conflicts may occur. Any conflict that occurs must be detected and resolved. Local consistency is enforced by pessimistic locking of the local replica of the database.

5.1.1 The Replication Module

In DeeDS, replication is handled by the replication module. The replication module interacts with the storage manager TDBM, and with the DeeDS Operating system Interface (DOI). TDBM is a store manager with support for nested transactions (Brachman & Neufeld 1992). The DOI is a layer added between the DeeDS database and the underlying hardware which makes it possible to make the DeeDS platform independent (Andler et al. 1998). Figure 5 depicts the replication module and how it interacts with the other parts of the DeeDS system.

![Figure 5: Replication in DeeDS (adapted from Matheis & Müssig (2003))](image)

The replication module logs all updates on data made during a local transaction. When the local transaction has been committed, the log is propagated to all other DeeDS nodes. The changes are then integrated into the remote replica(s). This is currently implemented
5.1 Replication in DeeDS

in DeeDS by using a number of modules: the logger, the propagator, the integrator, the version vector handler (VV handler) and the log filter (depicted in figure 5).

The logger is responsible for logging any changes made during a local transaction. When a transaction has been committed, the log is forwarded to the propagator module. The propagator distributes the update message containing the log(s) to all other replicas of the database. Currently, the log is distributed over TCP via point to point connections. A more efficient solution would be to use broadcast or multicast. On the remote nodes, the update message is received by the integrator module.

The integrator module consists of several submodules, these are: the receiver, the conflict detector, the conflict resolver and the updater. The receiver submodule is responsible for receiving update messages sent by propagators on other DeeDS nodes. The conflict detector then checks if the update made is in conflict with any earlier updates on this local node. This is done by using version vectors\(^2\) and the log filter. Three different types of conflicts may occur: write-write conflicts, read-write conflicts, and read-write cycles (Matheis & Müssig 2003). Matheis & Müssig (2003) describes conflict detection in DeeDS in more detail.

5.1.2 Local Conflict Detection & Resolution

If a (global) conflict is detected, the conflict resolver has to resolve the conflict. To achieve predictability, DeeDS detects and resolves conflicts locally, without communication between replicas over the network. If a conflict is detected, it is vital that it is resolved within a predictable amount of time in order to achieve predictable real-time database access, and in a deterministic way for global consistency. That is, if two or more replicas are in conflict, all replicas must resolve the conflict in the same way. DeeDS supports a set of simple generic replication policies. The chosen value may be:

- based on the new value resulting from an operation, e.g., mean, min or max value of two conflicting updates
- chosen from the highest prioritised replica
- based on timestamps (e.g., use the oldest or newest value)

If no conflict has been detected or when any detected conflict has been resolved, the updater writes the changes to the local copy of the database, and to the log filter. This must be done as one atomic action. During the integration process, the involved database objects and the log filter must be locked so that no other process manipulates them when the conflict detector is working. If the local database and the log filter are not kept mutually consistent, conflicts may be undetected with potential system failure as the result. Integration is done by using a special kind of transactions, integration transactions which run on a lower priority than regular database transactions. This makes sure that local access to the database does not need to wait an unpredictable amount of time for any integration processing. Thus, local real-time guarantees are kept.

To illustrate local conflict detection and resolution, a small example is used.

Example 1: Local Conflict Detection in DeeDS

Figure 6 depicts how a write-write conflict between two replicas can be resolved. In this case, two replicas, replica A and replica B, receives two simultaneous updates. Replica A receives an update of variable S, to 5 ($U : S = 5$), and replica B receives another update

\(^2\)For more details of the version vector algorithm used in DeeDS see Lundström (1997)
of $S$ to 3 ($U : S = 3$). Each replica commits the updates locally so that $S = 5$ on node A, and $S = 3$ on node B. Thus, the database is now globally inconsistent. After replica A has committed the update, it propagates the change to all other replicas in the system, so does replica B. Thus replica A will receive an update $U : S = 3$ which is in conflict with a previously committed update. Likewise, replica B will receive the update $U : S = 5$ which also is in conflict with a previously committed update. If we assume that a simple conflict resolution policy is used (e.g., use the maximum value) replica A will keep its value of $S$ (since $5 > 3$) while replica B will commit the new update ($U : S = 5$) again since $5 > 3$. Thus the conflict has been detected and resolved.

5.1.3 Key Design Choices in the DeeDS Architecture

One of the main aims of the DeeDS project is to enforce predictability. In order to do this a number of design choices have been made to eliminate unpredictable delays (Andler et al. 1998). The design choices are:

- the database resides in main-memory in order to avoid unpredictable disk access,
- the database is fully replicated in order to remove unpredictable network access, and
- updates are allowed to commit locally, on any node, and changes are propagated as soon as possible (eventual consistency) in order to avoid unpredictable distributed commit processing.

These design choices makes real-time database accesses in DeeDS predictable since all accesses are made in local main memory. Further, the database is guaranteed to be globally consistent eventually since all changes are propagated to other nodes, where conflicts are detected and resolved in a predictable and deterministic way.

5.2 The PRiDe Replication Protocol

The PRiDe replication protocol is similar to the replication scheme used in the DeeDS architecture. Like replication in DeeDS, PRiDe is an optimistic replication protocol where transactions are executed and committed locally. Eventual consistency is reached through use of the continuous-convergence protocol (Gustavsson & Andler 2005). As stated in Section 2.4, the continuous-convergence protocol is based on three notions (Gustavsson & Andler 2005):
5.2 The PRiDe Replication Protocol

- local consistency,
- local predictability, and
- eventual global consistency through continuous convergence

Continuous convergence means that “i) updates are continuously propagated and integrated, ii) conflicts are continuously and optimistically resolved, and iii) transactions are never rolled back; all conflicts are resolved by forward conflict resolution” (Gustavsson & Andler 2005).

5.2.1 Stable and Optimistic Versions

A fundamental difference from the DeeDS replication scheme is that PRiDe provides access to stable and optimistic versions of replicas. Transactions, initiated by an application, are not committed to the database replica directly (as they would be in DeeDS). Instead, their operations are regarded as tentative and are stored in the data structures used by PRiDe. Tentative operations may be in conflict with other tentative operations on other nodes. When all such conflicts has been detected and resolved for a specific tentative operation, it is considered to be stable, and is integrated in the database replica. The database thus represents the stable state of the replica, while the database together with the tentative operations represent the optimistic state of the replica (Section 2.4).

5.2.2 Generations & Conflict Sets

PRiDe uses two data structures to detect and resolve conflicts, generations and conflict sets. A generation is a vector of logically concurrent operations made to a replicated object. As updates in a generation are concurrent, a single generation can contain at most one update from each replica. Each entry in a generation either contains a null value, a no_update value or an update performed in the node in the vector entry. A conflict set for a replicated object ro is an ordered list of generations with operations to ro. Generations in the conflict set are ordered by their generation number, which is analogous to a logical timestamp. A generation with a lower number is processed before a generation with a higher number.

If a local replicated object receives an operation which is newer than the newest local generation for that replicated object, a new generation, and all generations in between are created on the local replica for the replicated object. When all generations have been created, the operation is added to the corresponding generation. If the local replica has not made any operation for the replicated object in the newly created generations, it inserts a no_update message in its entry in the generations.

When a generation is complete, i.e., when all entries in the generation vector are either no_update or update (i.e., all operations for this generation is known to the replica), the integration process can detect and resolve all conflicts permanently and the generation will be stabilised. When this is done, the now stable operation is included in the database.

The PRiDe protocol supports two kinds of read operations, stable reads and optimistic reads. Stable reads are executed directly on the stable state of replica, and ignores any tentative operations stored in the conflict sets. An optimistic read operation, optimistically resolves all conflicts among operations in the conflict set for the queried object, and applies the remaining operations to a copy of the stable version of the queried object. The read is then executed on the resulting optimistic state of copied object.

---

3 Logically concurrent means that the replication module cannot determine which operation was executed before the other. It does not imply that both operations happened at the exact same time.
5.2.3 Semantic Conflict Detection & Resolution

The PRiDe protocol, like DeeDS, uses an automatic, fully distributed conflict detection & resolution mechanism. Updates are collected into generations for each replicated object, where any updates within the same generation are regarded as syntactically conflicting. As generations stabilise, conflicts are detected and resolved. During conflict resolution, semantic information (if available) is used to determine if syntactically conflicting operations are in fact compatible. Resolution policies can be generic, such as the cake cutter method or the cheese cutter method, or application specific in order to make full use of semantic information.

5.2.4 Steps in PRiDe Replication

An application initiates a transaction which updates or reads one or more objects in the database. This results in that an operation arrives at the local replica. The operation is replicated to other nodes in the following way:

1. PRiDe considers it to be tentative and includes it into its data structures. The application can now continue its work.
2. The tentative operation is propagated to all other replicas of the database.
3. The tentative operation is added into the data structures on remote nodes, and tentative conflict resolution is performed, depending on the system configuration.
4. When a generation is complete, it is stabilised by performing permanent conflict resolution.
5. After a tentative operation has been stabilised, the operation is validated. For an (optimistic) read, it is checked if the originally read value corresponds with the final value. For an update, it is checked if the update was performed with the supplied parameters, or if it was somehow changed by a conflict resolution action. If the result from validation requires it, compensation is performed. Compensation allows conflict victims to perform compensating actions, such as resubmitting operations that were discarded during conflict resolution.

5.3 Replication in the TimesTen In-Memory Database

The fundamental motivation for replication in the TimesTen database system is to achieve high availability for data while minimising the impact on performance. Additional benefits include possibility to recover from failures (improved fault-tolerance), and the ability to perform on-line maintenance without downtime (The TimesTen Team 2004, Ch. 1). Figure 7 depicts a typical replication configuration in the TimesTen database.

As stated in section 2.5, replication in the TimesTen database system follows a master-subscriber pattern, where updates made at the master replica are copied onto one or more corresponding subscriber replicas (The TimesTen Team 2004, Ch. 1). Thus, TimesTen is a state-transfer system (Saito & Shapiro 2005).

4The cake cutter method discards updates from the conflict set until the resource requirements of all remaining updates can be satisfied (Gustavsson n.d.).

5The cheese cutter method modifies update parameters to reduce their resource requirements such that all requirements in the set can be satisfied (Gustavsson n.d.).
On each replica a replication agent controls replication. On a master replica, the replication agent reads the records from the transaction log and forwards any detected updates on the subscriber replica(s). On the subscriber replicas, the replication agent applies the update to the local replica (The TimesTen Team 2004, Ch. 1).

5.3.1 Replication Schemes

The master-subscriber pattern makes a number of different replication configurations possible (Oracle 2006b):

- Active–passive configurations,
- Active–active configurations, and,
- N–Way configurations.

The active–passive configurations is similar to the leader-follower replication scheme (see Section 2.2.1). One replica is configured to run as master (active), and the other replica(s) are running in passive mode. In an active–active and N–way configurations (similar to the multi-master replication scheme, see Section 2.2.1), the involved replicas are configured to run both as master and subscriber (Oracle 2006b); thus there are two replication agents running on each node, a master agent and a subscriber agent. An update made on one of the replicas will be copied onto all the corresponding subscribers. Actually, a N–way configuration is an active–active configuration with more than two replicas involved.

For this dissertation, an active–active configuration using asynchronous replication is interesting since this is the type of configuration that is most similar to the DeeDS architecture, and the most suitable one for use as a whiteboard architecture in a mobile environment.

5.3.2 Consistency Guarantees

Depending on the requirements of the application, the TimesTen database can be configured to support different levels of consistency guarantees. Updates can be replicated to the subscriber replica(s) in the following ways (The TimesTen Team 2004, Ch. 1) (cf. Section 2.2.2):
• asynchronous,
• synchronous (return receipt), and
• synchronous (return twosafe).

Asynchronous replication can be compared with the eventual consistency guarantee offered by the DeeDS architecture. When asynchronous replication is used, an application that makes an update to its local replica continues working. It does not wait for the update to be distributed and integrated on other replicas. However, in the DeeDS architecture, one knows that the database replicas will eventually converge into a consistent state. In the TimesTen database there is no such guarantee if asynchronous replication is used (The TimesTen Team 2004, Ch. 1).

When the application cannot tolerate asynchronous replication, synchronous replication can be used. In the TimesTen database, there are two flavours of synchronous replication: return receipt, and return twosafe. Basically, the return receipt option synchronises the master and subscriber replicas by blocking the application after commit on the master until the update has been received by the subscriber(s). However, there is no guarantee that the updates are integrated on the subscriber replica(s). The return twosafe option provides fully synchronous (cf. immediate) replication between the master and subscriber replicas. The application is blocked after it has issued a commit request to the master. Before the update is committed on the master, the update is distributed to the subscriber replica(s) and committed on them. Only then will the update be committed on the master and the application will be allowed to continue working (The TimesTen Team 2004, Ch. 1).

5.3.3 Conflict Detection & Resolution

The TimesTen database uses a syntactic (timestamp based) approach to detect conflicts. Each update is associated with a timestamp based on the local system clock, that is, different updates within one transaction will receive different timestamps. Conflicts are detected by comparing timestamps. If the timestamp of an update or insert is newer than the timestamp of the existing tuple, the existing tuple is updated. Conflicts are simply ignored, e.g., if the timestamps are equal, the update (or insert) is discarded, and most important, if the timestamp is older, the update (or insert) is also discarded. Thus, lost updates may be a problem. Another important issue with conflict detection and resolution in the TimesTen database is that read-write conflicts are not detected or resolved.

Another issue is that of how conflicts involving a delete operation are handled: delete/insert conflicts are not detected, delete/update conflicts are detected but they are not possible to resolve.

5.4 Replication in the Polyhedra RTRDB

As stated in Section 2.6, the Polyhedra RTRDB currently utilises the leader-follower replication scheme (i.e., it is a single-master system) in order to achieve fault-tolerance. Polyhedra implements fault-tolerance by running more than one replica of the database as a fault-tolerant pair (ENEA Polyhedra 2005b, Polyhedra 2005c). Figure 8 depicts such a fault-tolerant pair. One database runs as the master database, the other database(s) runs in standby mode. When the standby database(s) starts, it requests a snapshot from the master database. Client applications issue all updates to the master database, which feeds the standby database(s) with journal records to keep it up to date. Thus, after the transaction
has committed on the master replica, the passive replica(s) are updated immediately (see Section 2.6) (Polyhedra 2005a).

![Figure 8: Fault-tolerant configuration (adapted from ENEA Polyhedra (2005b))](image)

If the master replica fails, a standby replica will change its mode to master; client calls made to the database will transparently be directed to the new master replica. Polyhedra calls this a *fail-over* (Polyhedra 2005a, Polyhedra 2005c). This makes the Polyhedra RTRDB fault-tolerant. Two drawbacks with the leader-follower replication scheme are: *(i)* Updates can be performed only on the master replica, and *(ii)* all data are replicated on every passive replica. It would be advantageous if an application could perform updates on its local replica, and the replica itself would then be responsible for propagating the updates to all other replicas. Further, it would also be advantageous if a replica does not need to replicate all data (full replication), but instead only the data that the application accesses on the local replica (virtual full replication).
6 Comparisons

This section compares replication in the DeeDS architecture with the PRiDe replication protocol. A comparison is also made between DeeDS, DeeDS NG, TimesTen and the Polyhedra RTRDB.

6.1 Comparison between DeeDS and PRiDe

This section compares the replication scheme used in the DeeDS architecture with the PRiDe replication protocol. Both the replication scheme used in the DeeDS architecture and the PRiDe replication protocol are targeted towards real-time systems and thus predictability is important. For this reason, both replication schemes support local updates to a replica and global eventual consistency.

Although the two replication schemes share a number of features, there are some distinct differences. In the DeeDS architecture a transaction initiated by an application is performed in the local replica of the database directly. Hence, there is only an optimistic state of the database replicas. Any inconsistencies between replicas must be tolerated by the application that uses them. DeeDS is a state-transfer system, and conflict detection and resolution make use of syntactic information. This limits the ability to perform more tailored conflict resolution. Instead DeeDS supports some simple generic resolution policies (see Section 5.1.2).

<table>
<thead>
<tr>
<th></th>
<th>DeeDS</th>
<th>PRiDe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conflict detection &amp; resolution</td>
<td>Syntactic conflict detection and a set of simple generic conflict resolution policies</td>
<td>Syntactic and semantic conflict detection. A set of generic resolution policies and possibility to define application specific policies.</td>
</tr>
<tr>
<td>Operation replication</td>
<td>State-transfer</td>
<td>Operation transfer,</td>
</tr>
<tr>
<td>Replica versions</td>
<td>Optimistic state only</td>
<td>Stable and optimistic state</td>
</tr>
<tr>
<td>Operation execution</td>
<td>Operations are directly executed on the local database replica</td>
<td>Operations are kept in data structures until they are stabilised. Then they are integrated in the local database replica</td>
</tr>
</tbody>
</table>

Table 1: Comparison of replication in DeeDS and the PRiDe replication protocol

In PRiDe, an operation (resulting from a transaction) is at first regarded as tentative, and integrated in the generation and conflict set data structures. When all conflicts for the update has been permanently resolved it is regarded as stable and integrated in the local database replica. In this way PRiDe supports two versions of replicas, a stable and an optimistic version (Section 5.2.1). Applications that cannot tolerate inconsistencies can then use the stable versions, while applications that could benefit from using the optimistic state can do so. PRiDe is an operation-transfer system, and semantic information as well as syntactic information are used for detecting and resolving conflicts. If no semantic information is available, PRiDe supports the same kind of conflict detection and resolution as replication.
in DeeDS. PRiDe can also implement state-transfer operations by sending the new state as a parameter in an “overwrite” operation. Table 1 summarises the comparison.

### 6.2 Comparison of DeeDS, DeeDS NG, TimesTen and Polyhedra

The four systems differ in some essential aspects, such as the number of masters, replication strategy, and consistency guarantees. Table 2 summarises the comparison of the four systems. DeeDS NG is a future version of the DeeDS architecture and is included for completeness.

<table>
<thead>
<tr>
<th></th>
<th>DeeDS</th>
<th>DeeDS NG</th>
<th>TimesTen</th>
<th>Polyhedra</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Real-Time guarantees</strong></td>
<td>Hard real-time</td>
<td>Hard real-time</td>
<td>Soft real-time</td>
<td>Soft real-time</td>
</tr>
<tr>
<td><strong>Resides in</strong></td>
<td>Main memory</td>
<td>Main memory</td>
<td>Main memory</td>
<td>Main memory</td>
</tr>
<tr>
<td><strong>Replication scheme</strong></td>
<td>Multi-master (M = N)</td>
<td>Multi-master (M = N)</td>
<td>Active-passive/Active-active</td>
<td>Single-master (leader-follower)</td>
</tr>
<tr>
<td><strong>Data replication</strong></td>
<td>Full</td>
<td>Virtual Full</td>
<td>Full</td>
<td>Full</td>
</tr>
<tr>
<td><strong>Operation replication</strong></td>
<td>State transfer</td>
<td>Operation Transfer</td>
<td>State transfer</td>
<td>State transfer</td>
</tr>
<tr>
<td><strong>Scheduling of operations (ordering)</strong></td>
<td>Syntactic (version vectors)</td>
<td>Syntactic (conflict set which is ordered by using timestamps)</td>
<td>Syntactic (timestamps)</td>
<td>One transaction at a time, no pre-emption</td>
</tr>
<tr>
<td><strong>Consistency guarantees</strong></td>
<td>Global: Eventual, Local: Immediate</td>
<td>Global: Eventual, Local: Immediate</td>
<td>Asynchronous/Synchronous</td>
<td>Immediate</td>
</tr>
<tr>
<td><strong>Conflict detection</strong></td>
<td>Syntactic (version vectors and log filter)</td>
<td>Syntactic &amp; semantic (conflict set and generations)</td>
<td>Syntactic (timestamps)</td>
<td>No conflicts can occur</td>
</tr>
<tr>
<td><strong>Conflict resolution</strong></td>
<td>A set of simple generic policies</td>
<td>Both generic policies and application specific policies</td>
<td>Syntactic (timestamps)</td>
<td>No conflicts can occur</td>
</tr>
<tr>
<td><strong>Active functionality</strong></td>
<td>ECA Rules</td>
<td>ECA Rules</td>
<td>No</td>
<td>Active queries (SQL)</td>
</tr>
</tbody>
</table>

1The number of masters is equal to the number of nodes.

Table 2: Comparison of replication schemes

The DeeDS architecture is targeted towards large real-time systems. Thus even though its mechanisms are suitable for use as a whiteboard architecture in a MANET, it is not suitable when the mobile hosts have limited resources (e.g., as a PDA). However, the replication scheme used in DeeDS and the PRiDe replication protocol are both suitable for use for the type of scenario described in Section 2.7.

The replication model in the TimesTen database is lacking a number of features which are desirable for a whiteboard architecture (Section 2.5). While it does support writing to multiple masters (active-active), and a variant of eventual consistency (asynchronous replication), it does not detect read-write conflicts. Moreover, delete/insert conflicts are not
6.2 Comparison of DeeDS, DeeDS NG, TimesTen and Polyhedra

Comparison of DeeDS, DeeDS NG, TimesTen and Polyhedra

For these reasons, the TimesTen database is not suitable for use in the type of scenario described in Section 2.7.

The Polyhedra RTRDB currently only supports updates made to a single master replica, and immediate consistency is used. This is not sufficient to use instead of the replication scheme in the DeeDS architecture or PRiDe for the type of scenario described in Section 2.7. However, it seems possible to extend the Polyhedra RTRDB with support for multiple masters and eventual consistency. Full replication is already supported. Another advantage of using the Polyhedra RTRDB as a base, is that it is a small and fast embedded system, and thus seems suitable for use in scenarios as described in Section 2.7.

6.2.1 Conflict Types

In the DeeDS architecture and TimesTen database, different types of conflicts are considered. The DeeDS architecture considers write-write conflicts, read-write conflicts and read-write cycles. The TimesTen database on the other hand considers update conflicts, uniqueness conflicts and delete conflicts. Table 3 maps the different conflict types for the different systems together.

<table>
<thead>
<tr>
<th>DeeDS</th>
<th>TimesTen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write - Write conflicts</td>
<td>Update conflicts</td>
</tr>
<tr>
<td></td>
<td>Uniqueness conflicts</td>
</tr>
<tr>
<td></td>
<td>Delete conflicts</td>
</tr>
<tr>
<td>Read - Write conflicts</td>
<td>Not considered</td>
</tr>
<tr>
<td>Read - Write cycle conflicts</td>
<td>Not considered</td>
</tr>
</tbody>
</table>

Table 3: Conflict types in TimesTen and the DeeDS.
7 Extension Designs

As argued in section 6.2 all three systems have different aspects that makes them unsuitable for use as a whiteboard architecture for mobile embedded systems in a MANET. The Polyhedra RTRDB is lacking in specifically its replication scheme and consistency guarantees. It seems possible to use some of the features in the DeeDS architecture to alleviate this. The replication mechanism in the TimesTen database does not satisfy a number of important aspects, such as the ability to detect read-write conflicts.

This section presents the two design proposals for the MMR/EC (Multi-Master Replication with Eventual Consistency) extension. The first design proposal, presented in Section 7.1, is based on how replication is implemented in the DeeDS architecture. This design proposal is specific for the Polyhedra RTRDB. The second design proposal, presented in Section 7.2, is based on the PRiDe replication protocol. Although the design is presented by using the Polyhedra RTRDB, this proposal can be ported to another relational database system.

7.1 Based on DeeDS

In this design proposal support for multi-master replication and eventual consistency is implemented as a module on top of the RTRDB kernel. Events from the RTRDB is caught by using a specific Polyhedra API, namely the Journal API (England 2003).

7.1.1 Prerequisites for Multi-Master Replication

In order to implement the MMR/EC extension in the Polyhedra RTRDB, several requirements must be met. The following requirements have been identified:

1. **Logging**
   To be able to log changes made to the local database replica, the MMR/EC module must:
   
   (a) be notified when a transaction begins, commits or aborts
   (b) identify the data object changed by the local transaction
   (c) have knowledge of the new value, and preferably of the operation. If the operation is not available (i.e., we only know the after value), this limits the ability to perform “clever” conflict resolution policies.

2. **Propagation**
   To propagate the updates to other nodes:
   
   (a) each replica needs to be aware of all other replicas in the replicated system
   (b) a reliable communication protocol, e.g., a reliable broadcast must be available

3. **Integration**
   To integrate remote changes to the local database replica, the MMR/EC module must:
   
   (a) receive logs sent by other nodes over the network
   (b) ensure local consistency
   (c) schedule local updates and integration updates
7.1 Based on DeeDS

(d) integrate remote updates without triggering a new update. This requires that the replication module can discriminate between local updates and integration updates

(e) detect and resolve conflicts

7.1.2 High Level Design

Section 7.1.1 presents a number of prerequisites that must be met in order to extend the Polyhedra RTRDB with multi-master replication.

Logging In order to detect when a transaction begins, commits or aborts (prerequisite 1a) the Polyhedra Journal API could be used. This, however, is not currently implemented. The Journal API uses the callback programming model, and when a transaction either begins, commits or aborts, an appropriate callback function is invoked by the API.

With each data change, the Journal API makes information available which identifies the table, and the primary key column values of the updated data. This could be used in order to identify the data that was updated (prerequisite 1b). In this way, the new value (also referred to as after value) will be accessible (prerequisite 1c).

Propagation In order to reach consistency, an update on a replica must be propagated to all other replicas in the replicated system. If virtual full replication is used the update should be propagated to the replicas that store the segment of the database that was updated.

In the DeeDS architecture this is achieved by using reliable broadcast. This implies that a replica needs to be aware of the other replicas that should receive the update message and propagate it to them (prerequisite 2a). This can be solved by using a group management protocol. Nodes that want to be part of a replicated database joins the group, and leave when they should no longer be members.

Integration To be able to receive update messages sent over the network (prerequisite 3a), a process or thread (on each node which holds a replica of the database) should listen for incoming update messages.

In order to enforce local consistency, access to the local replica must somehow be synchronised. Further, to ensure that conflicts are detected and resolved correctly, the local database replica and the local log filter must be kept consistent (prerequisite 3b). This can be accomplished by locking the log filter (e.g., by using semaphores) while the database replica and the log filter is being updated. However, since the transactional model used in the Polyhedra RTRDB does not allow concurrent transactions and is non pre-emptive, no such locking is needed.

In the DeeDS architecture, integration is done by using integration transactions, which run on a lower priority than local transactions. In Polyhedra, it is currently not possible to use a second set of transactions. Thus regular (i.e., local) transactions must be used for integration purposes. This leaves two options: (i) local transactions are used for integration and a flag is set on them to represent integration transactions, or (ii) integration updates are written directly into the database without using the transactional layer. Further, if the transactional layer is used, the MMR/EC module must be able to discriminate between transaction events (signalled from the Journal API) from local transactions and from integration transactions (prerequisite 3d). Otherwise, an integration transaction would be propagated to other nodes, thereby creating integration cycles. Given that the MMR/EC
module can discriminate between local transactions and integration transactions, it is possible to integrate remote changes by starting an integration transaction (prerequisite 3d). Discriminating between integration transactions and local transactions is currently not supported by Polyhedra, and is thus a requirement on the Polyhedra system (see Section 7.1.3).

The MMR/EC module must be able to run the integration transaction, and thus needs to be able to schedule them and the local transactions using some scheduling policy (prerequisite 3c). Currently, local transactions are placed in a FIFO queue. One solution would be to simply add the integration transactions in the existing FIFO queue. Since we do not try to support hard real-time guarantees, this would be a feasible solution. A replica would not be blocked if another replica is temporarily unavailable since the either the information is already sent to the replica, or the replica is oblivious to the fact that such information exists.

As in the DeeDS system, conflicts are detected (prerequisite 3e) by using version vectors and the log filter. Generic conflict resolution is supported in the same way as in the DeeDS architecture (Section 5.1.2). Application specific policies are out of scope for this dissertation.

7.1.3 Requirements on Polyhedra

In order to implement the extension in the Polyhedra RTRDB, a number of requirements needs to be satisfied by the database.

- The Journal API must be implemented.
- There must be a way to integrate updates into the database, e.g., by using a special type of transactions (integration transactions) or by writing directly to the database without using transactions.
- There must be some way of discriminating between local transactions and integration transactions executed in the RTRDB kernel.
- If FIFO scheduling of integration transactions and local transactions is not feasible for a specific application, this issue must be addressed in the Polyhedra RTRDB.

7.1.4 Architectural Design

The multi-master replication extension will be added as a module on top of the Polyhedra RTRDB. The proposed design is depicted in Figure 9. Applications communicate directly with the RTRDB and are unaware of the MMR/EC module. The Journal API functions as a layer between the MMR/EC module and the RTRDB. It detects whenever a transaction is started, aborted or committed and invokes the appropriate callback function so that the MMR/EC module can take the appropriate action. Propagation is made at the MMR/EC module level according to the chosen propagation strategy. That is, if the broadcast propagation strategy is used, the propagator simply broadcasts the updated data over the network. When the MMR/EC module needs to integrate remote updates, it does so by manipulating the local database replica directly.

7.1.5 Replicating an Update Operation

To illustrate how the different modules are supposed to work (depicted in Figure 10), a small example is used. In this case, an operation which updates the value of data $s$ and sets it to 5 is used. Since Polyhedra only allows one transaction at the time to be executed, and
7.1 Based on DeeDS

Figure 9: Proposed design for MMR/EC support in the Polyhedra RTRDB, based on DeeDS

is non pre-emptive, the locking steps in the example are not required (see Section 7.1.2). However, they are included in the example for completeness.

Figure 10: Replication of an update operation

Example 2: Replicating an Update Operation
The first thing that happens is that a client issues an operation which sets $s = 5$ (step 1). The replication module detects that a new transaction has been started and creates a new log ($Log_T$) for this transaction (step 2). All updates made in $Log_T$ are added to the log (step 3). When the transaction commits, an end of transaction will be be caught and $Log_T$ will be be marked (in some way) as done. If the transaction is aborted instead of committed, the log should simply be removed (step 4). When the logger detects that $Log_T$ is done, it sends it to the propagator (step 5). The propagator distributes $Log_T$ as an update message to all other nodes that are in the replication group (step 6). When the integrator receives the update message, it locks the local log filter, and all database objects (in the local replica of the database) that were affected by the remote transaction on another replica (step 7). When
all necessary data have been locked, the conflict detector submodule checks for conflicts by comparing the log filter and the update message (this process uses version vectors). If a conflict is detected it is resolved locally (step 8). If no conflict was detected, or after the conflict has been resolved, the new updates are written to the log filter and the local database (step 9). When the changes are written, the locks are released (step 10).

7.2 Based on PRiDe

In this design proposal support for multi-master replication and eventual consistency is implemented as a layer on top of the RTRDB kernel. Figure 11 depicts the proposed design. This type of design is chosen because operations should not be included in the database until they are stable. Applications issue their operations to the MMR/EC layer. Tentative operations are then stored in the generation and conflict set data structures until they are stable. They are then integrated in the local database replica by the replication layer as part of the stabilisation process. An advantage of this approach is that the MMR/EC layer will have access to (i) the old value, (ii) the operation, and (iii) the operation parameters. From this information, the new value can also be derived. This allows for more elaborate conflict detection and resolution since semantic information is available.

Figure 11: Proposed design for MMR/EC support in the Polyhedra RTRDB, based on PRiDe

7.2.1 Object Level Interface

Since the PRiDe replication protocol is designed to work with objects and operations on objects, the applications interface to the MMR/EC layer will be on an object level. If the application were to use a relational interface, such as ODBC, the MMR/EC layer would need to implement its own SQL parser to identify the data objects concerned by the SQL statement. The choice and implications of an object- or relational-level interface are discussed in Section 8.3.

The interface between applications and the MMR/EC layer supplies a number of operations for the applications to access the database:
7.2 Based on PRiDe

- Create - Used to create a new database object. The application needs to supply the object identifier (OID) for the object to be created, along with all the attributes that should be set for the object.

- Destroy - Destroy a database object. It is sufficient for the application to supply the OID for the object that should be destroyed.

- Fetch - Read a database object. The application needs to supply the OID and the read method that the application wishes to invoke, e.g., a read method can be specified to return attribute \( a_1 \), and another read method can be specified to return attribute \( a_2 \). Further, the type of the read operation also needs to be specified, i.e., is it a stable read or an optimistic read.

- Update - Update a database object. The application needs to supply the OID, the update method and the parameters that are to be updated.

The interface could also support configuration of properties of data, such as data residency (e.g., main-memory or disk), propagation strategies (e.g., ASAP or bounded). Further, it should be possible to specify application specific conflict resolution policies and to associate them with either a class or an object.

7.2.2 Mapping from Object Level to Relational Level

Since Polyhedra is a relational database, it is necessary to map the new stable value, resulting from the stable operation, down to the relational model when they are to be integrated in the local database replica. This mapping is done in the following way:

- A class maps to a table
- An attribute maps to a column in a table
- An instance maps to a row in a table
- The object identifier (OID) maps to the super-key for a row (e.g., table and primary key)

An operation invoked by a transaction are then translated into a query for the underlying database, e.g., in the case of Polyhedra an ODBC or JDBC query can be used.

7.2.3 Moving Requirements

By placing the MMR/EC extension as a layer between the application and the underlying database, requirements are moved from the underlying database onto the replication module. This makes it easier to implement the MMR/EC layer on top of another database. This will be discussed more in Section 8.1.
8 Discussion

8.1 Portability

The design proposal based on the DeeDS architecture (Section 7.1) requires instrumentation of the underlying database (e.g., detect start and commit of a transaction). This limits the portability of the replication module since such instrumentation often is specific for different database systems (e.g., Polyhedra uses a specific API, the Journal API, which is not available in other database systems). The design proposal based on the PRiDe replication protocol (Section 7.2) on the other hand, requires no instrumentation of the underlying database. All requirements that were previously placed on the database systems can now be handled within the replication module instead. What is needed is a mapping from the object level used in PRiDe to the relational level used in a relational database system. This increases the portability of the replication module since it is easier to port such a mapping which is based on standardised SQL, rather than using proprietary API's.

8.2 Advantages & Disadvantages of the Design Proposals

The two design proposals have different advantages and disadvantages. These are summarised in Table 4:

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Based on DeeDS</th>
<th>Based on PRiDe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applications can communicate with the Polyhedra database without knowledge of the replication module. The regular API's can still be used.</td>
<td>No instrumentation of the underlying database system is required. Requirements are moved from the database onto the replication layer. This increases portability.</td>
<td></td>
</tr>
<tr>
<td>No updates can be made without being detected by the replication module</td>
<td>Semantic information can be used to reduce the amount of conflicts in the system. More elaborate conflict resolution is possible.</td>
<td></td>
</tr>
<tr>
<td>Disadvantages</td>
<td>Specific to the Polyhedra database. Not easily ported to other systems since Polyhedra specific API's are used.</td>
<td>Updates can be made without using the API (not detected by the replication module)</td>
</tr>
<tr>
<td>Only semantic conflict detection and resolution is supported.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Advantages & disadvantages for the two design proposals.

8.3 Application - MMR/EC Interface

In the current design proposal based on PRiDe, the interface between applications and the replication layer will be on an object level (Section 7.2.1). There are two reasons for this:
(i) PRiDe is designed to work with objects, and (ii) if the application sends relational operations (e.g., SQL), the replication layer would need to parse these in order to determine which data objects that are affected by the operation. Since the main focus of this dissertation is not the intricacies of SQL parsing, an object level interface was chosen. However, it is an interesting question what type of interface, object or relational, that is most suited for applications. Further study of this topic is proposed as future work (Section 9.4).

Other interesting questions regarding the interface between applications and the replication module is: (i) how should a database administrator configure application specific conflict resolution policies, (ii) where should data reside, and finally (iii) how should the database schema be modified. Should the application interface support functionality for this so that also a schema change is replicated to other nodes? Studying this is also proposed as future (Section 9.4).

8.4 Propagation Strategies

In the current extension design and in the DeeDS architecture reliable broadcast is used to propagate update messages to other replicas in the system. Another class of propagation strategies is epidemic propagation (Demers, Greene, Houser, Irish, Larson, Shenker, Stur- gis, Swinehart & Terry 1988, Khelil, Becker, Tian & Rothermel 2002). Saito & Shapiro (2005, p. 46) state:

Epidemic propagation lets any two sites that happen to communicate exchange their local operations as well as operations they received from a third site—an operation spreads like a virus does among humans.

Since the communicating participants in the MANET (as described in Section 2.7) at a given time is unlikely to be able to communicate with all other nodes, an epidemic propagation strategy would provide a good fit. Replicas that are not able to communicate with each other directly, would still be able to exchange operations via other replicas. Since replicas may be disconnected from the network an arbitrarily amount of time, the epidemic approach needs to make sure that such replicas receive all operations when they return to the network.
9 Conclusions

This section concludes the dissertation. The dissertation is briefly summarised, the main contributions are identified, related work is discussed and future work is proposed.

9.1 Summary

Section 2.7 describes a scenario in which mobile participants (in this case rescue workers) are cooperating in a distributed environment. The participants are equipped with some type of hand-held device (e.g., a PDA), which they use to exchange information (e.g., locations of survivors, dangerous object, exits and so on). Tatomir & Rothkrantz (2005) argue that a shared whiteboard architecture can be used to implement a communication architecture for this kind of scenario. However, since the hand-held devices (from now on referred to as nodes) can join and leave the network at any time, and be disconnected from the network for an arbitrarily amount of time, implementing such a shared whiteboard architecture is hard. Brohede & Andler (2002) argues that a distributed, real-time database could be used to implement a shared whiteboard architecture useful for complex sharing applications such as distributed real-time simulations. In this dissertation it is argued that such a database system using a multi-master replication scheme, using full replication and eventual consistency, could also be used to implement a shared whiteboard architecture for the scenario described above. The DeeDS architecture, as well as the PRiDe replication protocol, both provides replication schemes that are suitable for such a whiteboard architecture. Thus, they are used as guidelines for two design proposals for how to extend the Polyhedra RTRDB with support for multi-master replication using full replication and eventual consistency. Also the TimesTen database system has a replication scheme with features that are suitable for use in the type of whiteboard architecture discussed above. However, some features are lacking (see Section 6.2), which makes the TimesTen replication scheme insufficient for use in a shared whiteboard architecture used for communication for the type of scenarios described above.

As mentioned above, this dissertation presents two design proposals for how to implement support for multi-master replication using full replication and eventual consistency in a main-memory relational DBMS. The first design proposal is based on the replication scheme used in the DeeDS architecture (Section 7.1). This design proposal is specific for the Polyhedra RTRDB since it is designed to be implemented as a module which makes use of events that occur within the database kernel. In order to implement the design proposal, a number of prerequisites and requirements need to be met by the underlying database (Sections 7.1.1 and 7.1.3).

The second design proposal is based on the PRiDe replication protocol (Section 7.2). Although this design proposal is designed for the Polyhedra RTRDB, it is a general design, which can be ported to other systems. This is because in this design proposal, support for the MMR/EC extension is implemented as a layer on top of the underlying database. Rather than communicating directly with the database, applications issue their operations to the MMR/EC layer, which after any conflicts have been detected and resolved, integrates the operations in the database. In this way, requirements can be moved from the database and instead be placed in the MMR/EC layer (Section 7.2.3). This makes it easier to move the MMR/EC implementation onto another database system (Section 8.1).
9.2 Contributions

The main contributions made by this dissertation are presented in the list below:

- A design for how to implement the replication scheme used in the DeeDS architecture in the Polyhedra RTRDB has been proposed.
  - A number of prerequisites and requirements on the underlying database have been identified.
- A portable design for how to implement the PRiDe replication protocol in a relational database has been proposed.
  - An interface between applications and the MMR/EC layer has been described.
  - A mapping between the object level used in the MMR/EC layer and the relational data model used in the underlying database has been described.

9.3 Related Work

Brohede & Andler (2002) argues that a distributed active real-time database system (a DARTDBS) could be used to implement a whiteboard architecture which could be used for communication in for instance complex real-time simulations. Further, in Brohede & Andler (2005) they argue that such a DARTBS could be used as infrastructure for information fusion applications. In this dissertation it is argued that such a database system could also be used for communication between mobile nodes in a MANET.

Fahl & Risch (1997) presents an approach to object view management which could be used for transparently working with relational data as if it was stored in an object-oriented database. This kind of technology could be used to implement the mapping from the object-level used in PRiDe to the relational level of for instance the Polyhedra RTRDB or the TimesTen database. The focus of their paper is on query processing rather than update operations which also needs to be considered.

9.4 Future Work

This section proposes future work.

- Proof of Concept Implementations
  This dissertation presents two design proposals. The most obvious future work proposal is to make a proof of concept implementation for each design proposal. To be able to make a proof of concept implementation for the first design proposal, the requirements on the Polyhedra RTRDB must be fulfilled.

- Experimental Performance Impact Analysis
  Using the proof of concept implementations, experiments should be carried out in order to measure the performance impact of the two different MMR/EC extensions. Several measurements are interesting: throughput, response time, storage requirements and network load. Without knowledge of how the different MMR/EC extensions affect the performance of the system, it is hard to tell whether they are useful or not.

- Application Interface
Object or Relational Level
A more thorough investigation about the interface between applications and the MMR/EC layer would be interesting. Particularly the following questions could be answered: should the application interface be on object level or relational level? If relational level is used, how does this affect the MMR/EC layer?

Functionality
What type of functionality should be supported in the application interface. Besides create, destroy, read and update operations (see Section 7.2.1) used by the application programmer, how should the database designer configure application specific conflict resolution policies and where data resides? Further, should the database schema also be modified using the interface?

- Referential Integrity
In the second design proposal (see Section 7.2) objects are mapped down relations in a relational database. Future work needs to investigate how referential integrity between objects should be enforced. As an example, consider a relationship between an object and several subobjects. If for example two subobjects are updated on different nodes, the object which consists of the subobjects, may be incorrect.

- Bounded Delay Replication
Section 2.3 briefly mentions that the DeeDS architecture supports two classes of replication: (i) best effort replication (ASAP), and (ii) bounded delay replication. The design proposals contributed by this dissertation have only considered the best effort approach. If applications would benefit from support for bounded delay replication, it would be useful to include support for this in the MMR/EC extension.
References


