# Requirements Analysis Document

Database Team

**PAID Project**

**Team**

Informatik XII

WS 1998/1999

Technische Universität München

February 5, 1999

**Revision History:**

Version R0.1 01/11/99 First Preview, Johannes Schmid
Version R0.2 01/13/99 Updated - now in StarOffice, Johannes Schmid
Version R0.3 01/14/99 Added Requirements, Johannes Schmid
Version R0.4 01/16/99 Merged with John's Use Cases and Florian's Sequence Diagrams, John Feist, Florian Schönherr, Johannes Schmid
Version R0.5 01/18/99 Some minor changes, Johannes Schmid

**Preface:**

This document addresses the requirements of the PAID system. The intended audience for this document are the designers and the clients of the project.

**Target Audience:**

Client, Developers

**PAID Members:**

| | |
|---|---|
| **Project Management:** | Bernd Bruegge, Guenter Teubner |
| **Team Coaches:** | Ralph Acker, Stefan Riss, Ingo Schneider, Oliver Schnier, Anton Tichatschek, Marko Werner |
| **Architecture Team:** | Asa MacWilliams, Michael Luber |
| **Authentication & Security Team:** | Klaas Hermanns, Thomas Hertz, Guido Kraus, Gregor Schraegle, Tobias Weishaeupl, Alexander Zeilner |
| **Database Team:** | Osman Durrani, John Feist, Florian Klaschka, Johannes Schmid, Florian Schoenherr, Ender Tortop |
| **Learning Team:** | Burkhard Fischer, Juergen Knauth, Andreas Loehr, Marcus Toennis, Martin Uhl, Bernhard Zaun |
| **Network & Event Service Team:** | Henning Burdack, Joerg Dolak, Johannes Gramsch, Fabian Loschek, Dietmar Matzke, Christian Sandor |
| **StarNetwork Integration & User Interface Team:** | Daniel Stodden, Igor Chernyavskiy, Inaki Sainz de Murieta, Istvan Nagy, Stefan Krause, Stefan Oprea |
| **Testbed Team:** | Bekim Bajraktari, Bert van Heukelom, Florian Michahelles, Goetz Bock, Michael Winter, Sameer Hafez |

**MILESTONES**

1/11/99 Release of RAD Template
1/14/99 Team RAD
1/18/99 Integrated RAD

## Table of Contents

# 1 General Goals

The purpose of the *PAID Database Subsystem* is to provide efficient access to the *Aftersales Database* and persistent storage. This is to be achieved by replication of the database to the worldwide DaimlerChrysler sales organization.

This includes dividing the aftersales data into replicable database subsets, remote query invocation if a database query cannot be answered locally, query caching and management of other data, such as user data.

# 2 Current System

This part intentionally left out. It will be filled by the documentation team.

# 3 Proposed System

## 3.1 Overview

The *Database Subsystem* can store all aftersales data - or just some specific subsets of it - locally on any JDBC capable database. It will provide its services to all other PAID subsystems and will be running on any kind of computer - starting at the main PAID servers at DaimlerChrysler headquarters down to the small PAID server which is running on a laptop being used as "mobile garage". Interactions with the Database subsystem will not involve actual users, but will instead be orchestrated by various other PAID subsystems.

The data stored locally must be up-to-date. This is achieved by replicating the changes of the main database down the PAID server hierarchy. For the requesting PAID subsystem there will be no difference between locally answered queries and queries answered via network. The network structure is hierarchical and thus potentially unlimitedly deep. Queries posed to the database subsystem, which cannot be answered locally, are passed on to the next server in hierarchy.

Changes the user does on the local data (i.e. changes in the Electronic Parts Catalogue) are replicated back, so all other PAID Servers will have them available within a short period of time.

All database queries are authenticated and secured using the methods provided by the *Security & Authentication Team*. Query caching and database subset replication is done in cooperation with the *Learning Team*.

## 3.2 Functional Requirements

### 3.2.1 Main purpose

The main purpose is to provide efficient local storage for the aftersales data and the *PAID Persistent Storage*.

The queries handled by the *Database Subsystem* will be in form of a string and not a like a prepared statement (including variables and symbols).

The lookup of data subsets will also be supported. The availability of data will be checked. The removal and replacement of subsets must be done in response of incoming requests.

### 3.2.2 Installation and replication

The system will allow installation of a base set of data from CD or other large-capacity medium.

The system will allow clients to subscribe to new releases of a subset (update); the update will be automatically broadcasted downwards the PAID network tree. If a client is not reachable, the update will be saved and the client will be notified later (via network). It then can decide when to request the update.

Updates can also be imported from CD or other large-capacity medias. Synchronization on demand of data on client and server computers will be possible, too.

The system will allow clients to do necessary updates on the server, which will be replicated up the PAID network tree.

### 3.2.3 Intelligent caching system

An intelligent and flexible caching system can be implemented, which enables the user to set the amount of the data cache in the memory and on the local hard drives. The system must use this cache as effectively as possible so that the least necessary data will be deleted to make place for the actual data.

### 3.2.4 Operability as a standalone system

The system must be operable when disconnected from network. All data replicated to the local database must be accessible. This is extremely necessary for a 'mobile garage'.

### 3.2.5 Data flow

The system will answer data requests either from local data or from remote data receiving via network. The decision will be made on base of the knowledge which data is locally stored. The data received from the network will be cached.

The system will allow data manipulation of local and/or remote data.

## 3.3 Nonfunctional Requirements

### 3.3.1 User Interface and Human Factors

The *Database Subsystem* is largely an internal subsystem of the PAID project. This subsystem will have no interaction with the user and will be at all times at least one step away from user commands.

The only concern in this case is language-dependence of the actual data in the database. However, this is mainly *StarNetwork / UserInterface Subsystem's* concern.

### 3.3.2 Zero administration

PAID Servers can run with so-called zero administration: no special administrative effort is needed for installation and keeping the system running. The system is capable of automatic administration after some necessary parameters were entered during installation. However, to reach optimal performance, some administration is needed, e.g. which subsets of the *Aftersales Database* should be replicated.

### 3.3.3 Documentation

All uses of the *Database Subsystem* will occur through a well-defined API. Only this API will be used by other PAID subsystems. Because of this, the *Database Subsystem's* API functionality will be documented using JavaDoc comments and UML models.

User documentation may be provided if issues arise in regards to installation of the database applications on Dealer server machines or other issues related to Hardware and other considerations. This will be provided to the *Documentation Team* and will be put into the Users Manual for the PAID project.

### 3.3.4 Hardware Consideration

There are several sets of hardware considerations that need to be mentioned in association with the *Database Subsystem*. First of all, the servers which will hold many subsets of the *Aftersales Database* will need to be powerful and scalable in order to supply the necessary short response time now - and in future. Typically, data replication and *Persistent Storage* for the PAID subsystems will need a certain amount of space to be stored. This may be a challenge if we consider a handheld device as a platform for PAID.

### 3.3.5 Performance Characteristics

The *Database Subsystem* definitely has performance constraints. Queries for aftersales data need to be very fast. Typically users will not want to wait more than a half minute for data to be displayed. Given this constraint, hardware and software should be optimized to allow such speedy access times.

The *Database Subsystem* relies on the *Networking* and other subsystems for communication with other system objects. This may result bottlenecks for the *Database Subsystem*. These bottlenecks will be dealt with by the other subsystem groups. Generally speaking, the sum total of all the performance constraints of all PAID subsystems for a particular task should be "reasonable".

This means, a local query should be answered below a second in average; remote queries should be answered below half a minute in average.

### 3.3.6 System Interfacing and location transparency

The *Database Subsystem* should be developed by the database team as an integral part of a larger framework composed of 5 other major subsystems. The system thus formed is named as PAID. The *Database Subsystem* itself does not offer any User Interface to the outside world, instead it receives inputs from other PAID subsystems.

The Data known to the  is divided into two main categories: on the one hand, the remote data, which is the set of all data not residing on the local database, but can be found remotely on parent servers. On the other hand, there is local data, which represents the set of all data present locally on the system, for example on hard disk, CD rom, floppy disk or local cache.

The PAID *Database Subsystem* should make its Data and services available to requests coming from anywhere in the PAID network. The PAID network consists of a tree-like structure made up of various servers. The requests for Data, that the *Database Subsystem* cannot locate locally, is searched in servers located higher in the hierarchical structure of PAID servers. The *Database Subsystem* must present a unified view of the Data to the other subsystems.

### 3.3.7 System Modifications

The *Database Subsystem* is free to be changed as long as its behavior stays the same and it still conforms to the specified APIs. The nature of the *Aftersales Database* can possibly change in the future. The *Database Subsystem* must develop a set of APIs which allow it to incorporate all these changes transparently to the other PAID subsystems.

### 3.3.8 Physical Environment and resource Issues

System resources (hard drive, system memory, etc.) should be adequate to handle the amount of data stored locally on the PAID system and the network traffic being processed by the PAID system.

### 3.3.9 Security Issues

All data which is transmitted via Network or which is located on CD or local store must be secured by the *Authentication & Security Subsystem* and only be accessible via an appropriate password. The database which is used must provide an encryption technique if encryption on local hard disk is needed.

All queries are authenticated and checked by the *Authentication & Security Subsystem*, before they reach the *Database Subsystem*. In order to guaranty the integrity and security of data, no other PAID subsystem will be allowed to run direct queries on any piece of data stored by the *Database Subsystem*.

### 3.3.10 Data management

To divide the aftersales data into subscribeable and updateable parts, Data Subsets are needed. Subset updates are needed, which can bring a specific subset from one version to the next version. This ensures the ability of replicating specific parts of the *Aftersales Database* to any number of client PAID systems. Besides that, there must be a possibility to add a specific subset of data to the local database, which can be updated later using the previously mentioned 'subset updates'.

### 3.3.11 Error Handling and quality issues

All the possible errors should be handled properly. Due to the impossibility of listing all the possible errors before the implementation they will be documented during the implementation phase.

All errors which occur in the *Database Subsystem* are either handled adequately by itself, or are adequately trapped and passed to the PAID system that initiated the transaction. In the extreme case that data cannot be stored on the local storage mechanism, the *Database Subsystem* must still provide access to aftersales data and upload capabilities by interfacing directly with a server through the *Network Subsystem*.

In case of some error occurring during data transfers, the system should have the capability to ensure the integrity of the incoming data as well as the updated data. No possible error should be able to corrupt the existing data set.

## 3.4 Constraints

The entire system must be written in 100% pure Java. Development will be done using the Together/J CASE tool. Any JDBC capable database will be supported. Source code control will be handled using CVS.

## 3.5 System Model

### 3.5.1 Scenarios

#### 3.5.1.1 Installation

A new installation of PAID is taking place. The *Database Subsystem* receives a request to install a base set of data from the CD to the *Local Aftersales Database*. This base set is necessary to guarantee the specified functionality of all PAID subsystems - no actual aftersales data is in the *Local Aftersales Database* yet. After user credentials are verified using the *Security & Authentication Subsystem*, the *Database Subsystem* installs this data. The next (optional) step is now to install some aftersales data locally.

#### 3.5.1.2 Install subsets of aftersales data from CD

The *Database Subsystem* receives a request to install a specific subset from CD. The subset is read from the media, then is decrypted and authenticated by the *Authentication & Security Subsystem* and then is inserted in the *Local Aftersales Database*.

#### 3.5.1.3 Install subsets of aftersales data from Network

The *Database Subsystem* receives a request to install a specific subset from Network. The subset definition must first be created on a parent PAID server by its *Database Subsystem*. The subset then is transferred back to the local system and then is inserted in the *Local Aftersales Database*.

#### 3.5.1.4 A subsystem access data

A PAID Subsystem requests access to a specific subset of the Aftersales Database through one of the Database Subsystem data classes. This request is equivalent to a query about some group of data. The *Database Subsystem* receives a database query already validated by the S*ecurity & Authentication Subsystem* and determines whether this request can be answered locally using the *Local Aftersales Database*. This is not the case, therefore the local *Query Cache* is checked if the query was answered recently using the Network so it could be answered without network traffic. The query could not be found in the *Query Cache*, so it is sent to the PAID server one step higher in hierarchy. It is answered there and transferred back to the requesting subsystem.

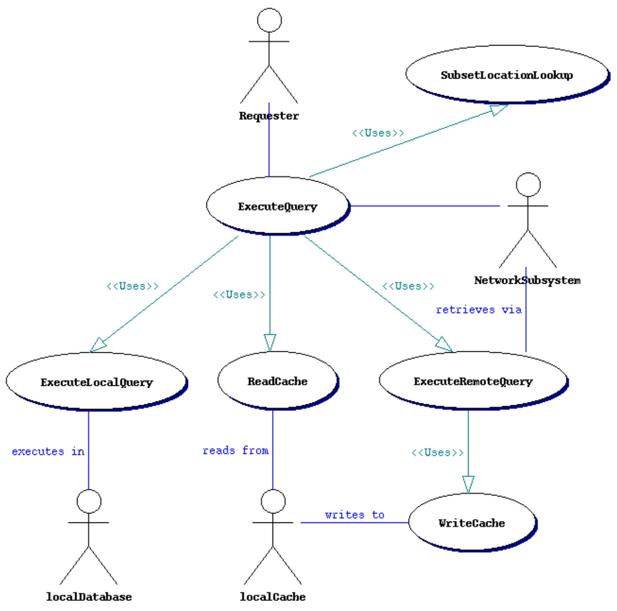
#### 3.5.1.5 A subsystem requests a data manipulation

A PAID Subsystem requests to change data on the aftersales database. The *Security & Authentication Subsystem* already validated this request. The *Database Subsystem* receives the query and then writes the updated data to the *Local Aftersales Database*, invalidates the *Query Cache* and attempts to upload this data to the parent PAID server. This request does not succeed, because the dealer is not currently connected to the network. The *Database Subsystem* then schedules this upload with the Event subsystem. As soon as the Dealer is online again, this update request is uploaded to the parent PAID server.

## 3.5.2 Use Case Models

### 3.5.2.1 UseCase diagram Retrieve Data

These Use Cases are needed to retrieve data within a given subset and provide the location transparency. The requester has to know which subset it wants to access.



### Use Case ExecuteQuery

| | |
|---|---|
| **Entry Condition** | Any Paid Subsystem requests data from a given subset, the action is already authenticated |
| **Flow of Events** | 1. Use SubsetLocationLookup to decide whether the requested data is locally available or not<br>2. Respectively use ExecuteLocalQuery or ReadCache<br>3. If the Cache does not have the data use ExecuteRemoteQuery |
| **Exit Condition** | Data has been retrieved or an error condition has occurred |
| **Special Requirements** | The local database must be alive |

### Use Case SubsetLocationLookup

see UseCase Diagram „Extract Subset“.

## Use Case ExecuteLocalQuery

| | |
|---|---|
| **Entry Condition** | ExecuteQuery requests locally available data |
| **Flow of Events** | 1. Send SQL-Query to local Database<br>2. Return ResultSet or SQL Error |
| **Exit Condition** | Data has been retrieved or an error condition has occurred |
| **Special Requirements** | The local database must be alive, otherwise an error is returned |

## Use Case ExecuteRemoteQuery

| | |
|---|---|
| **Entry Condition** | ExecuteQuery requests locally unavailable uncached data |
| **Flow of Events** | 1. Tell Network subsystem to retrieve requested data<br>2. On success: use WriteCache to cache the ResultSet locally<br>3. Return ResultSet or an error |
| **Exit Condition** | Data has been retrieved or an error condition has occurred |
| **Special Requirements** | The network subsystem should be alive, otherwise an error is returned |

## Use Case ReadCache

| | |
|---|---|
| **Entry Condition** | ExecuteQuery requests in local database unavailable data |
| **Flow of Events** | 1. Ask local Cache if it has the requested data<br>2. Return ResultSet or an error indication that the local Cache does not have the data |
| **Exit Condition** | Data has been retrieved or an error condition has occurred |
| **Special Requirements** | The local Cache must be alive, otherwise an error is returned |

## Use Case WriteCache

| | |
|---|---|
| **Entry Condition** | ExecuteRemoteQuery wants to update local cache contents |
| **Flow of Events** | Put the data in the local Cache |
| **Exit Condition** | Data has been cached or not (e.g. due to space limits) |
| **Special Requirements** | The Cache must be be alive, otherwise an error is returned there has to be enough space available locally |

## Actor Requester

This primary actor may be any Paid subsystem that needs data of a subset. But: The action must be authenticated!

## Actor NetworkSubsystem

This actor is either primary or secondary: The Network subsystem is secondary actor if ExecuteRemoteQuery wants to retrieve data from a parent server. Therefore Network has to send the request to the "parent Network subsystem". If the "parent network subsystem" receives the request for a query it represents a primary actor and uses ExecuteQuery to answer the request.

## Actor localCache

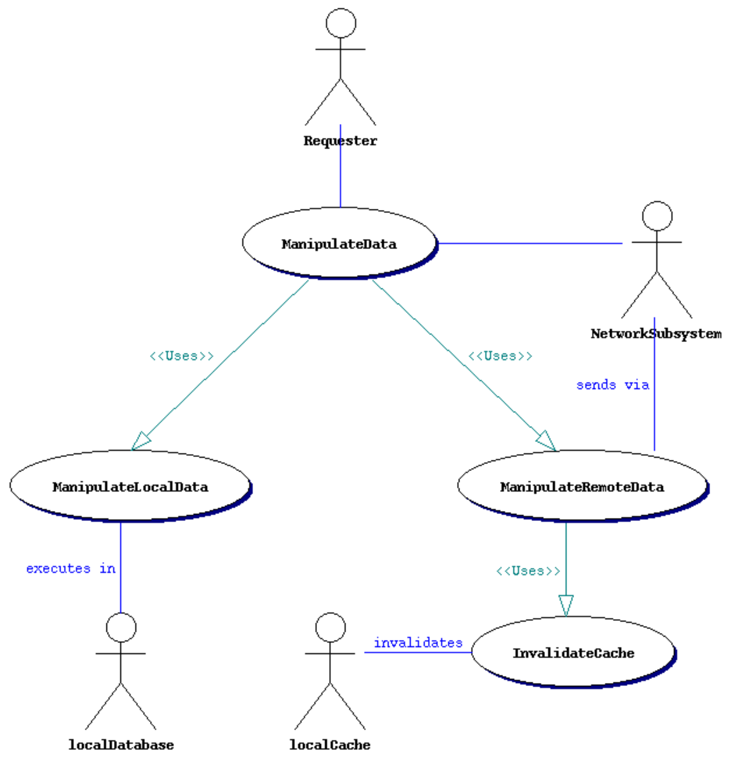
This secondary actor represents the local cache that is used by ReadCache and WriteCache.

## Actor localDatabase

This secondary actor represents the local database and is only used by ExecuteLocalQuery.

### 3.5.2.2 UseCase diagram Manipulate Data

These Use Cases are needed to manipulate data within a given subset and to ensure that changes are replicated to parent servers.



#### Use Case ManipulateData

| | |
|---|---|
| **Entry Condition** | Any Paid Subsystem wants to manipulate data, the action is already authenticated |
| **Flow of Events** | 1. Decide whether to manipulate locally, remotely or both<br>2. Use ManipulateLocalData in order to locally manipulate data<br>3. Use ManipulateRemoteData in order to remotely manipulate data |
| **Exit Condition** | Data has been updated or an error condition has occurred |
| **Special Requirements** | The local database should be alive |

#### Use Case ManipulateLocalData

| | |
|---|---|
| **Entry Condition** | ManipulateData wants to update local data |
| **Flow of Events** | 1. Send SQL-Statement to local Database<br>2. Return nothing or SQL Error |
| **Exit Condition** | Data has been updated locally or an error condition has occurred |
| **Special Requirements** | The local database must be alive, otherwise an error is returned |

## Use Case ManipulateRemoteData

| | |
|---|---|
| **Entry Condition** | ManipulateData wants to manipulate remote data |
| **Flow of Events** | 1. Tell Network Subsystem to send manipulation request to parent server to call ManipulateData on parent server<br>2. Use InvalidateCache to remove the manipulated data from the local cache<br>3. Return nothing or Error |
| **Exit Condition** | Data has been manipulated remotely or an error condition has occurred |
| **Special Requirements** | The network subsystem should be alive, otherwise an error is returned |

## Use Case InvalidateCache

| | |
|---|---|
| **Entry Condition** | ManipulateRemoteData wants to invalidate local cache contents |
| **Flow of Events** | Remove the data from the local Cache |
| **Exit Condition** | Data has been removed from the Cache |
| **Special Requirements** | The Cache must be be alive, otherwise an error is returned |

## Actor Requester

This primary actor my be any Paid subsystem that wants to manipulate data, but the action must be already authenticated!

## Actor NetworkSubsystem

This actor can be primary and secondary: It is secondary if ManipulateRemoteData requests a remote data manipulation on the parent server via the Network Subsystem. In the parent server this actor is a primary one if the Network subsystem receives the request for data manipulation from a client server. It first has to ask Authentication for ok and then uses ManipulateData in order to proceed the request.

## Actor localCache

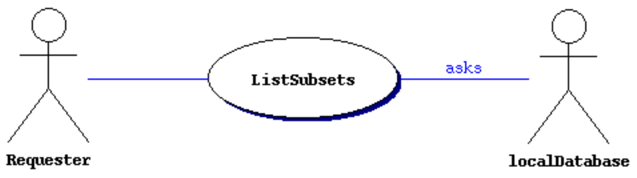
This secondary actor represents the local cache that stores recently executed queries and the results, therefore the cache entries that belong to a specific query must be removed if the data is updated.

### Actor localDatabase

This secondary actor represents the local database and is only used by ManipulateLocalData in this context.

### 3.5.2.3 UseCase diagram List Subsets

This use case is needed to retrieve a list of available subsets (locally or remote).



### Use Case ListSubsets

| | |
|---|---|
| **Entry Condition** | Any Paid subsystem wants to have a list of available subsets. |
| **Flow of Events** | Ask localDatabase for a list of available subsets |
| **Exit Condition** | List successfully retrieved or an error condition has occurred |
| **Special Requirements** | None |

### Actor Requester

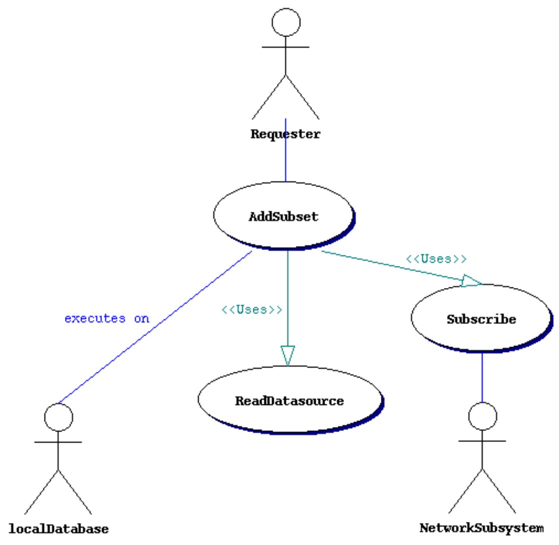
This primary actor represents a Paid subsystem that wants to know about the available subsets.

### Actor localDatabase

This secondary actor represents the local database that stores information about available subsets.

## 3.5.2.4 UseCase diagram Add Subset

These Use Cases are needed to add a subset to the local db.



### Use Case AddSubset

| | |
|---|---|
| **Entry Condition** | Learning or User Interface Subsystem wants to add a subset to the locally available ones |
| **Flow of Events** | 1. Decide where to retrieve the subset from (Network or a local source like CD/DVD)<br>2. Use ReadDatasource to retrieve the subset<br>3. Insert SubSet in the local Database<br>4. Use Subscribe for automatic receipt of future updates |
| **Exit Condition** | Subset successfully retrieved and inserted or an error condition has occurred |
| **Special Requirements** | Local database and the Datasource must be alive |

### Use Case ReadDatasource

see UseCase Diagram „Read Datasource“.

### Use Case Subscribe

| | |
|---|---|
| **Entry Condition** | AddSubset wants to subscribe to a recently requested Subset |
| **Flow of Events** | Tell Network subsystem to send our parent server the request for updates |
| **Exit Condition** | Subset successfully subscribed or an error condition has occurred |
| **Special Requirements** | The Network subsystem should be alive |

### Actor Requester

Here the possible requesters normally are the User Interface and the Learning subsystem.

### Actor NetworkSubsystem

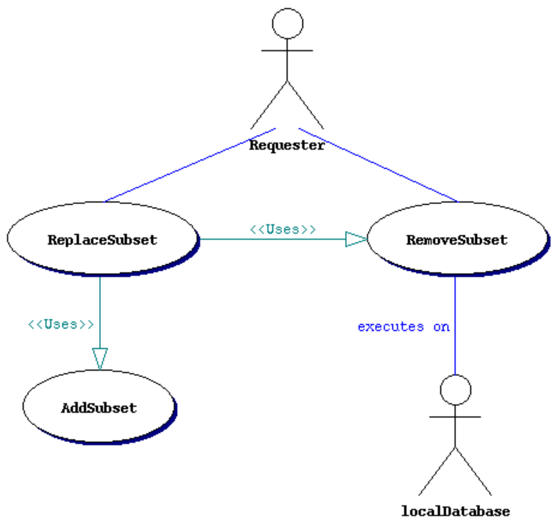
The Network Subsystem is used for the subscription of a subset.

### Actor localDatabase

This secondary actor represents the local database and is only used to insert a subset.

## 3.5.2.5 UseCase diagram Replace and Remove Subset

These Use Cases are needed to remove or replace subsets from/in the local db.



### Use Case ReplaceSubset

| | |
|---|---|
| **Entry Condition** | Learning or User Subsystem wants to replace a subset in the local database |
| **Flow of Events** | 1. Use RemoveSubset<br>2. Use AddSubSet |
| **Exit Condition** | Subset successfully removed and inserted or an error condition has occurred |
| **Special Requirements** | The subset should be in the local database |

### Use Case RemoveSubset

| | |
|---|---|
| **Entry Condition** | ReplaceSubset or a paid subsystem wants to remove a Subset from the local database |
| **Flow of Events** | Remove the subset from the database |
| **Exit Condition** | Subset successfully removed or an error condition has occurred |
| **Special Requirements** | The local database should be available otherwise an error is returned |

### Use Case AddSubset

See UseCase Diagram „Add Subset“.

### Actor Requester

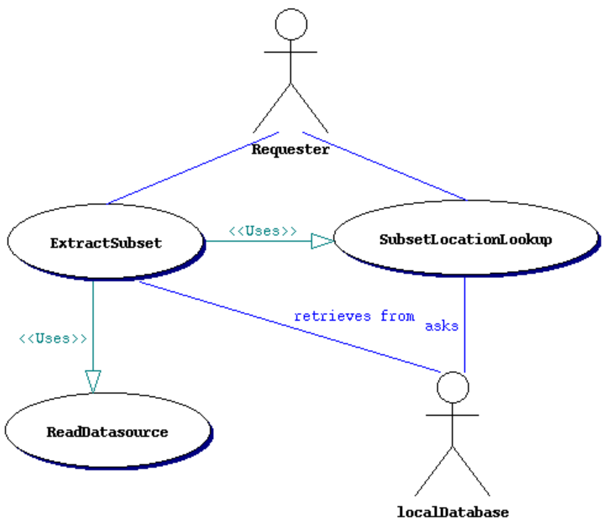
This primary actor represents the Learning, Network or User Interface subsystem.

### Actor localDatabase

This secondary actor represents the local database.

## 3.5.2.6 UseCase diagram Extract Subset

These Use Cases are needed to create subsets and to retrieve information about subsets.



## Use Case ExtractSubset

| | |
|---|---|
| **Entry Condition** | A subsystem wants to extract a subset for a subscriber |
| **Flow of Events** | 1. Use SubsetLocationLookup<br>2. If the subset is available in the local database, retrieve it<br>3. Otherwise use ReadDatasource to retrieve it<br>4. Return subset or error |
| **Exit Condition** | Subset successfully extracted/retrieved or an error condition has occurred |
| **Special Requirements** | Either local database or datasource must be alive |

## Use Case ReadDatasource

see UseCase Diagram „Read Datasource“.

## Use Case SubsetLocationLookup

| | |
|---|---|
| **Entry Condition** | A Paid subsystem wants to have location information of a specific subset |
| **Flow of Events** | 1. Ask the local db for the location of the specified subset<br>2. Return "locally available" or "not ..." or error |
| **Exit Condition** | The location information of the subset is returned or an error condition has occurred |
| **Special Requirements** | Local database must be alive |

## Actor Requester

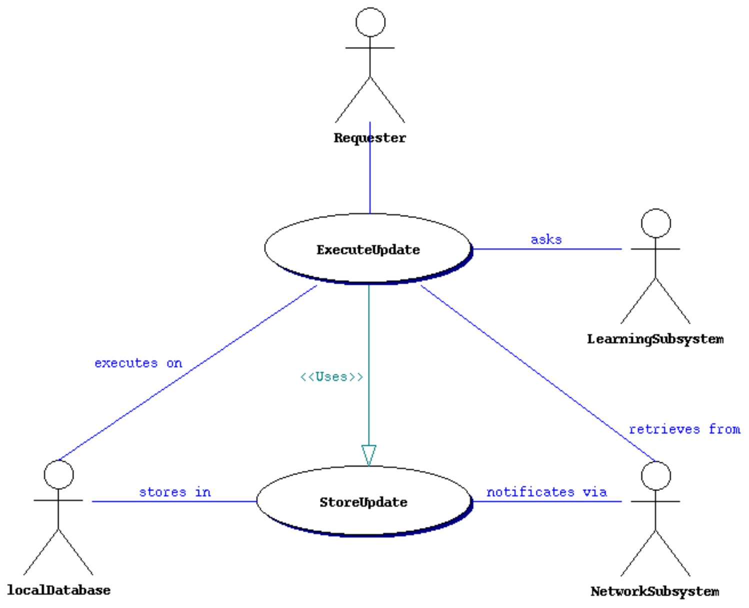
This primary actor may be the database subsystem or any other Paid subsystem.

## Actor localDatabase

This secondary actor represents the local database that hold informations about the location of subsets and some subsets themselves.

### 3.5.2.7 UseCase diagram Execute Update

These Use Cases are needed to apply and store Updates. Updates may be contain complete subsets "from scratch" or just smaller changes to the database.



### Use Case ExecuteUpdate

| | |
|---|---|
| **Entry Condition** | Learning or Network Subsystem wants to execute an update, the action is already authenticated |
| **Flow of Events** | 1. Get the specified update from the parent server via the Network Subsystem<br>2. Ask Learning whether to store the update or not<br>3. Execute the update on the local Database<br>4. Use StoreUpdate in order to locally store the update |
| **Exit Condition** | Update has been executed and stored or an error condition has occurred |
| **Special Requirements** | The local database should be alive |

### Use Case StoreUpdate

| | |
|---|---|
| **Entry Condition** | ExecuteUpdate wants to store an Update |
| **Flow of Events** | 1. Store the Update in the local Database<br>2. Tell Network Subsystem to send notification to subscribers |
| **Exit Condition** | Update has been stored locally or an error condition has occurred |

| | |
|---|---|
| **Entry Condition** | ExecuteUpdate wants to store an Update |
| **Special Requirements** | The network subsystem and the local database should be alive, otherwise an error is returned |

### Actor Requester

This primary actor may be the Learning or the Network subsystem, but the action must be already authenticated.

### Actor LearningSubsystem

This secondary actor represents the Learning subsystem that is needed to decide whether to store an Update locally or not.

### Actor NetworkSubsystem

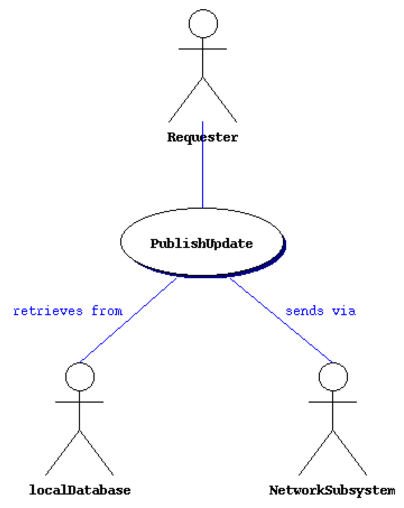
This secondary actor represents the Network subsystem that notificates the subscribers of updates or retrieves an update from the parent server.

### Actor localDatabase

This secondary actor represents the local database that stores data that has to be updated and stores Updates for subscribing clients.

## 3.5.2.8 UseCase diagram Publish Update

These UseCases provide the possibility to publish updates to subscribers.



## Use Case PublishUpdate

| | |
|---|---|
| **Entry Condition** | A Learning subsystem wants to push or pull updates to a list of subscribers. |
| **Flow of Events** | 1. Get the local stored update from the database<br>2. Send the update to the list of subscribers via the Network subsystem |
| **Exit Condition** | Update retrieved successfully or an error condition has occurred |
| **Special Requirements** | The requested update must have been stored previously in the local database and the local database must be available otherwise an error is returned |

## Actor Requester

This primary actor represents either the local Learning subsystem, that wants to push updates to subscribers, or a client Learning subsystem that wants to pull updates.

## Actor NetworkSubsystem

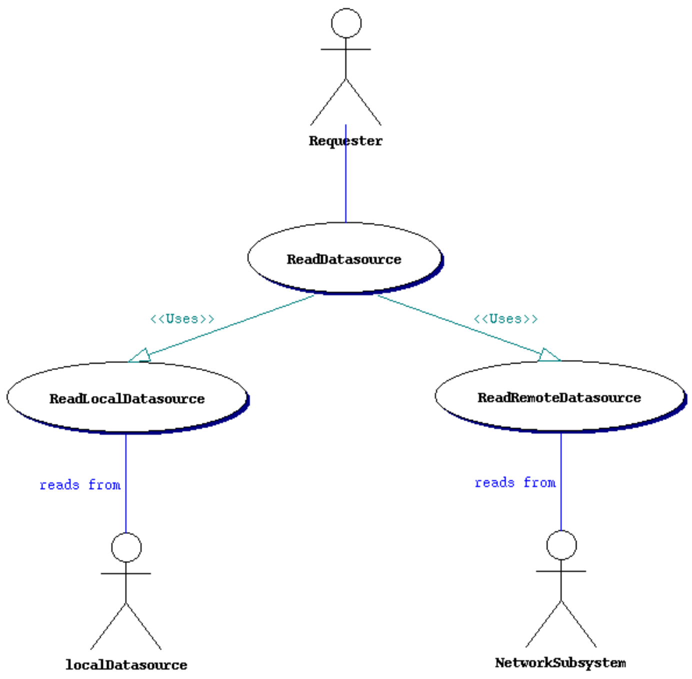
This secondary actor represents the network subsystem that is needed to send the update to the client.

## Actor localDatabase

This secondary actor represents the local database that stores the updates.

### 3.5.2.9 UseCase diagram Read Datasource

These use cases are needed to read a subset from a datasource like CD/DVD or Network.



#### Use Case ReadDatasource

| | |
|---|---|
| **Entry Condition** | AddSubset or ExtractSubset wants to read a subset from a datasource |
| **Flow of Events** | 1. Use either ReadLocalDatasource or ReadRemoteDatasource<br>2. Return subset or error |
| **Exit Condition** | Subset successfully retrieved or an error condition has occurred |
| **Special Requirements** | None |

#### Use Case ReadLocalDatasource

| | |
|---|---|
| **Entry Condition** | ReadDatasource wants to read a subset from a local datasource |
| **Flow of Events** | Read Datasource and retrieve subset |
| **Exit Condition** | Subset successfully retrieved or an error condition has occurred |
| **Special Requirements** | None |

## Use Case ReadRemoteDatasource

| | |
|---|---|
| **Entry Condition** | ReadDatasource wants to read a subset from the network |
| **Flow of Events** | Tell Network subsystem to retrieve the subset from the parent server |
| **Exit Condition** | Subset successfully retrieved or an error condition has occurred |
| **Special Requirements** | None |

## Actor Requester

This primary actor represents one of the other use cases: AddSubset or ExtractSubset.

## Actor NetworkSubsystem

This secondary actor represents the network subsystem as a remote datasource. The system has to forward the request to the parent server. The network system in the parent server then uses ExtractSubset.

## Actor localDatasource

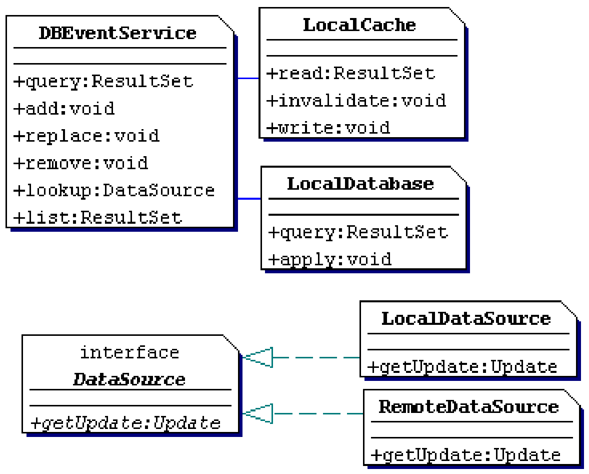
This secondary actor represents a local datasource like a CD, DVD...

## 3.5.3 Object Models

### 3.5.3.1 Data Dictionary

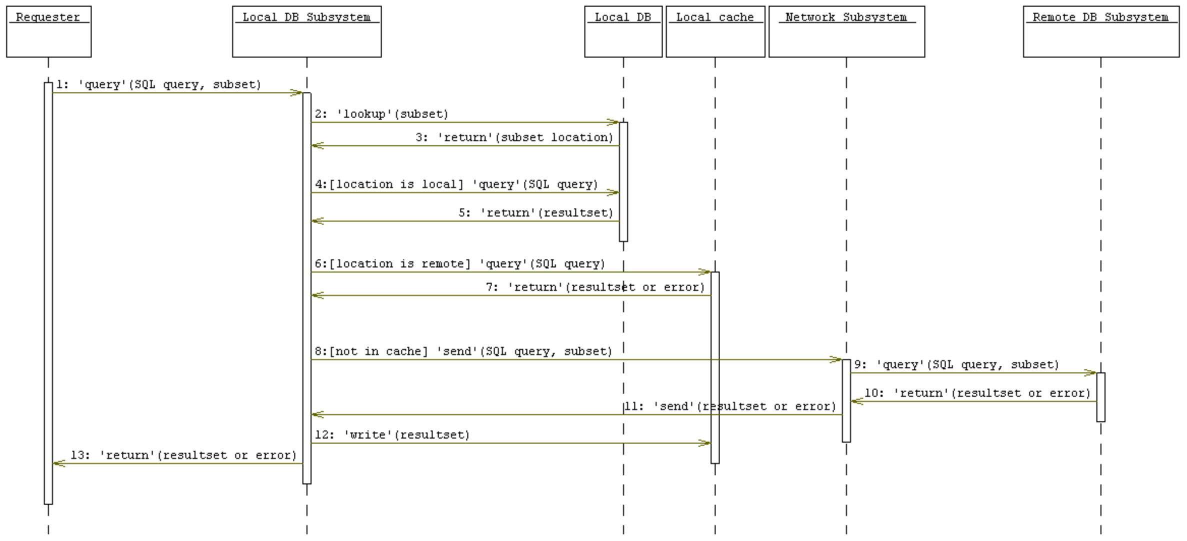
| | |
|---|---|
| **DB EventService** | This will be the main class of the database API. It receives events from Network Subsystem (the event bus) and reacts on them |
| **Local Cache** | This class represents the cache table of the local database |
| **Local Database** | This class acts as an interface for all activity on the local database |
| **Datasource** | An interface to generalize the access to multiple data sources |
| **Local Datasource** | Implements Datasource and represents some local media, for example the installation CD |
| **Remote Datasource** | Acts as an interface to Network Subsystem for update requests |

### 3.5.3.2 Class Diagrams



## 3.5.4 Dynamic Models

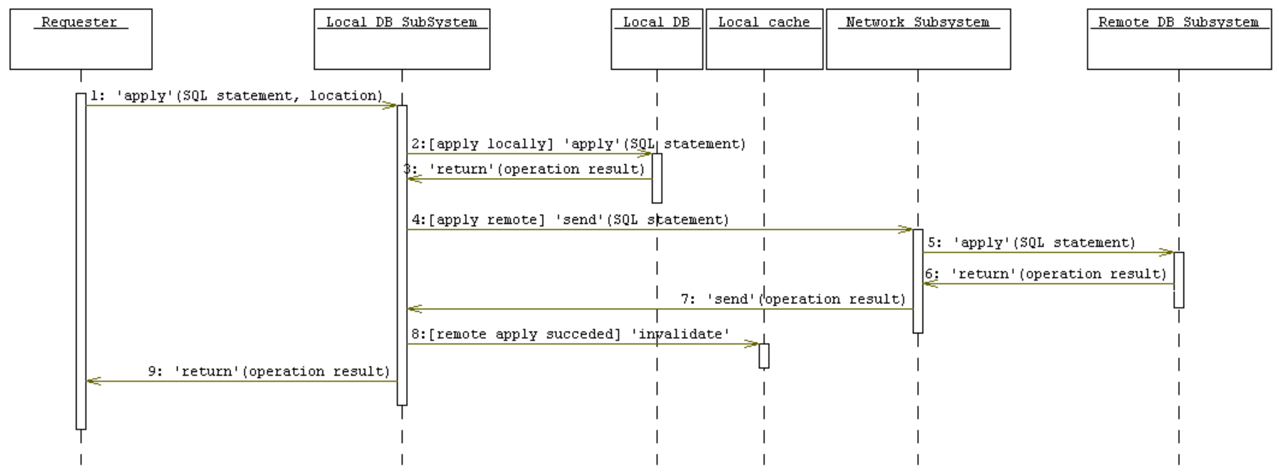
### 3.5.4.1 Retrieve Data


Requester
Local DB Subsystem
Local DB
Local cache
Network Subsystem
Remote DB Subsystem
1: 'query'(SQL query, subset)
2: 'lookup'(subset)
3: 'return'(subset location)
4:[location is local] 'query'(SQL query)
5: 'return'(resultset)
6:[location is remote] 'query'(SQL query)
7: 'return'(resultset or error)
8:[not in cache] 'send'(SQL query, subset)
9: 'query'(SQL query, subset)
10: 'return'(resultset or error)
11: 'send'(resultset or error)
12: 'write'(resultset)
13: 'return'(resultset or error)

Requester sends an SQL statement along with the subset to work on

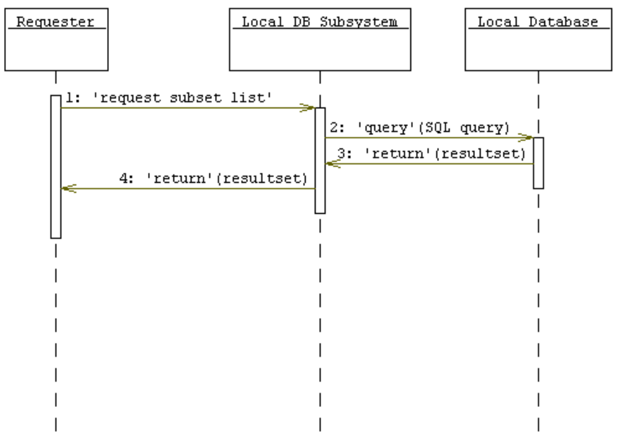
1. Local DB Subsystem looks up the subset's location from the lookup table in Local Database
2. Local DB Subsystem receives 'local' or 'remote' as the subset's location
3. if location is 'local' the SQL query is executed on Local Database
4. Local Database gives back the ResultSet
5. if location is 'remote', Local DB Subsystem searches for this query's ResultSet in Local Cache
6. Local Cache returns ResultSet if it was already cached, or an error if not
7. if the ResultSet has not been found in Local Cache, the SQL query is send to the parent server via Network Subsystem
8. Remote DB Subsystem receives the SQL statement and executes it itself
9. Remote DB Subsystem sends ResultSet or error via Network Subsystem
10. Local DB Subsystem receives ResultSet or error
11. if ResultSet was received from Remote DB Subsystem, it is written to Local Cache
12. Local DB Subsystem returns ResultSet or error

### 3.5.4.2 Manipulate Data


Requester
Local DB SubSystem
Local DB
Local cache
Network Subsystem
Remote DB Subsystem
1: 'apply'(SQL statement, location)
2:[apply locally] 'apply'(SQL statement)
3: 'return'(operation result)
4:[apply remote] 'send'(SQL statement)
5: 'apply'(SQL statement)
6: 'return'(operation result)
7: 'send'(operation result)
8:[remote apply succeded] 'invalidate'
9: 'return'(operation result)

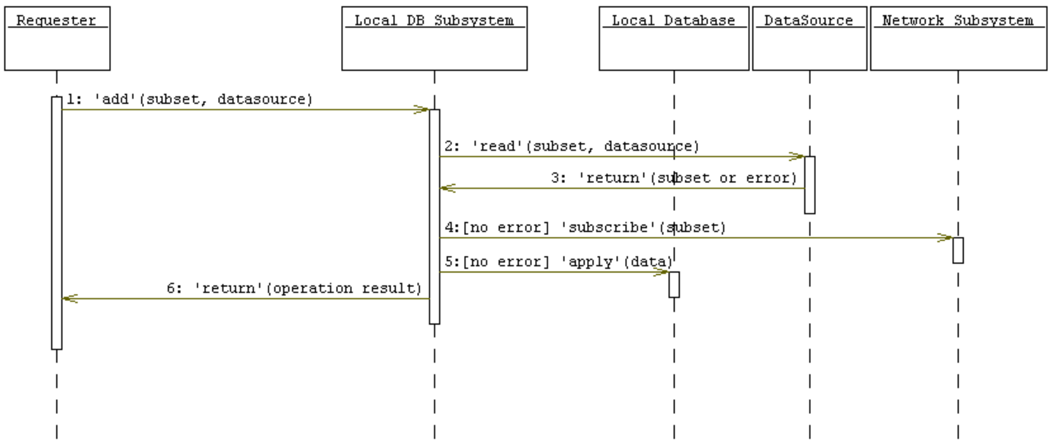
1. Requester calls Local DB Subsystem to apply an SQL statement on a given location (local, remote or both)
2. if the location is 'local', the statement is executed on Local Database
3. Local Database gives back confirmation or error
4. if the location is 'remote', the apply command is sent via Network Subsystem to Remote DB Subsystem
5. Remote DB Subsystem is called by Network Subsystem to apply the statement
6. the Remote DB Subsystem sends confirmation or error via Network Subsystem
7. Local DB Subsystem receives confirmation or error
8. if no error occurred, Local Cache is invalidated
9. Local DB Subsystem gives back confirmation or error

### 3.5.4.3 List subsets



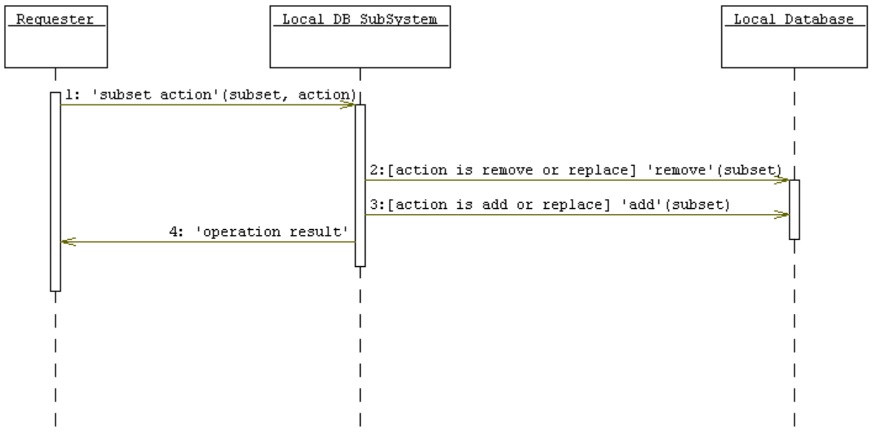
1. Requester requests subset list from Local DB Subsystem
2. Local DB Subsystem executes SQL query for all subset identifiers on Local Database
3. Local DB Subsystem receives ResultSet
4. Local DB Subsystem returns ResultSet

### 3.5.4.4 Add subset


Requester
Local DB Subsystem
Local Database
DataSource
Network Subsystem
1: 'add'(subset, datasource)
2: 'read'(subset, datasource)
3: 'return'(subset or error)
4:[no error] 'subscribe'(subset)
5:[no error] 'apply'(data)
6: 'return'(operation result)

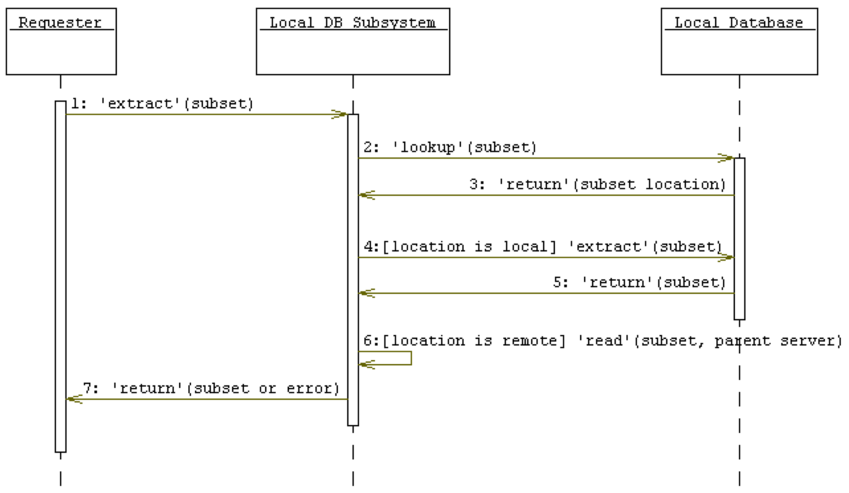
1. Requester calls add and gives the wanted subset and the datasource to read from
2. Local DB Subsystem executes 'read' on the given datasource (see Use Case ReadDatasource)
3. Datasource returns subset or error
4. if no error occurred, Local DB Subsystem subscribes via Network Subsystem for all updates on this subset
5. if no error occurred, Local DB Subsystem applies the retrieved data on Local Database
6. Local DB Subsystem returns the operation's result (confirmation or error)

### 3.5.4.5 Replace and remove subset


Requester
Local DB SubSystem
Local Database
1: 'subset action'(subset, action)
2:[action is remove or replace] 'remove'(subset)
3:[action is add or replace] 'add'(subset)
4: 'operation result'

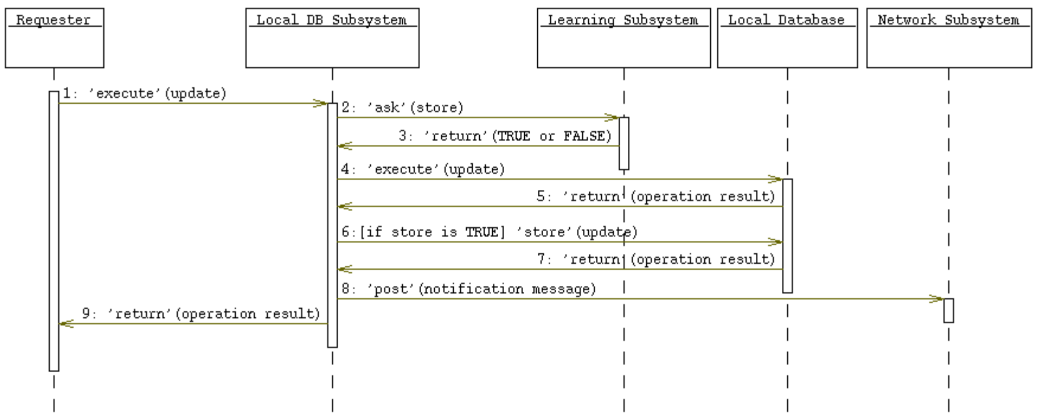
1. Requester calls 'request action' for a specific subset (replace for two subsets)
2. if the action is 'remove' or 'replace', remove the (old) subset from Local Database
3. if the action is 'add' or 'replace', add the (new) subset (see Use Case AddSubset)
4. return confirmation or error

### 3.5.4.6 Extract subset


Requester
Local DB Subsystem
Local Database
1: 'extract'(subset)
2: 'lookup'(subset)
3: 'return'(subset location)
4:[location is local] 'extract'(subset)
5: 'return'(subset)
6:[location is remote] 'read'(subset, parent server)
7: 'return'(subset or error)

1. Requester calls 'extract' and gives wanted subset identifier
2. Local DB Subsystem looks up the subset's location in the lookup table of Local Database
3. Local Database returns 'remote' or 'local' as the subset's location
4. if location is 'local', the subset data is extracted from Local Database
5. Local database returns the wanted subset data
6. if location is 'remote', read the subset from the parent server (see Use Case ReadDataSource)
7. Local DB subsystem returns the subset data or an error (if location is 'remote')

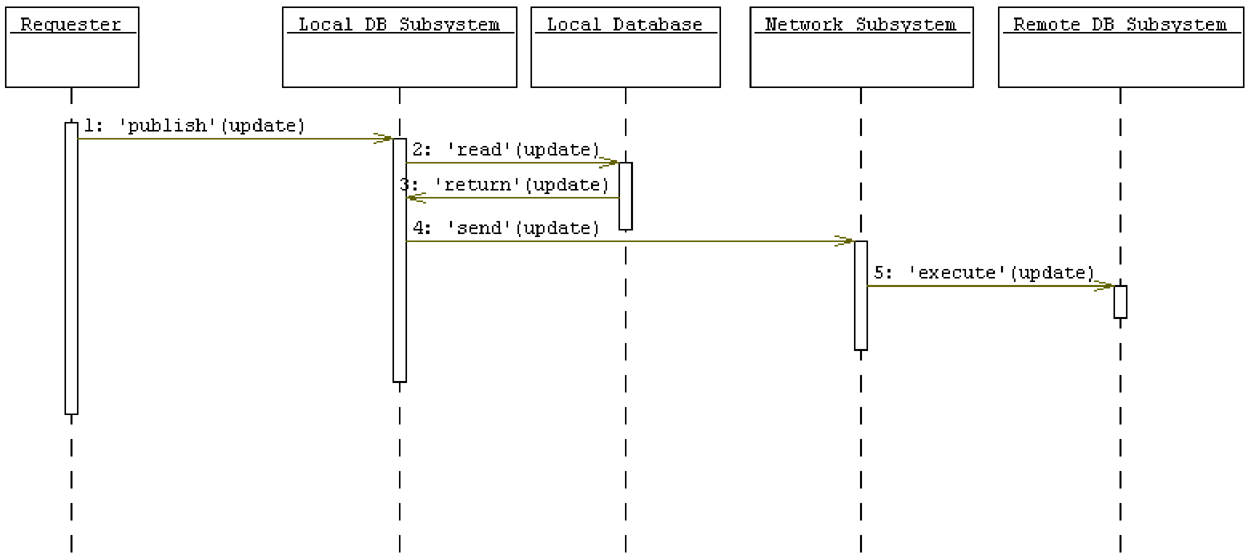
### 3.5.4.7 Execute update


Requester
Local DB Subsystem
Learning Subsystem
Local Database
Network Subsystem
1: 'execute'(update)
2: 'ask'(store)
3: 'return'(TRUE or FALSE)
4: 'execute'(update)
5: 'return'(operation result)
6:[if store is TRUE] 'store'(update)
7: 'return'(operation result)
8: 'post'(notification message)
9: 'return'(operation result)

1. Requester calls 'execute' along with the update data
2. Local DB Subsystem asks Learning Subsystem if the update data should be stored in the local update queue
3. Learning subsystem answers 'TRUE' or 'FALSE'
4. the update is executed on Local Database
5. Local Database returns operation result

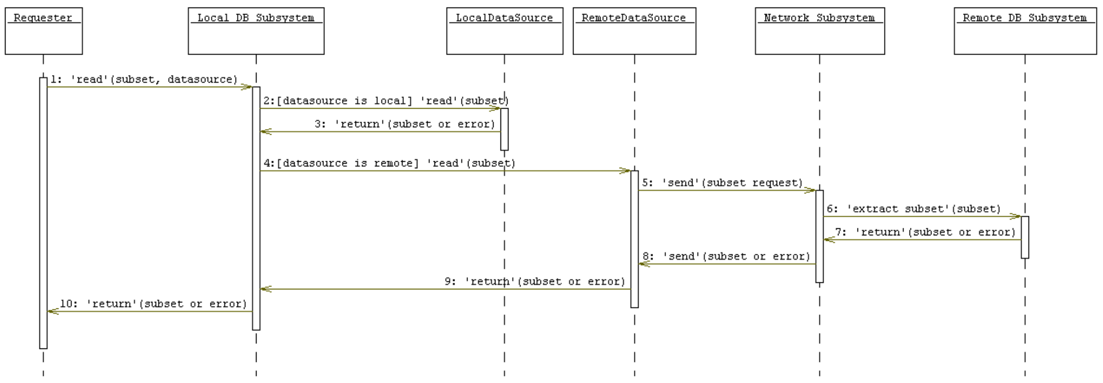
6. if store is 'TRUE', the update data is stored to the local update queue
7. Local Database returns operation result
8. Local DB Subsystem post a notification message on the event bus via Network Subsystem
9. Local DB Subsystem returns operation result

### 3.5.4.8 Publish update



1. Requester calls 'publish' along with the update identifier
2. Local DB Subsystem reads update data from Local Database
3. Local Database returns update data
4. Local DB Subsystem sends update data via Network Subsystem
5. Remote DB Subsystem receives command to execute the update data (see Use Case ExecuteUpdate)

### 3.5.4.9 Read DataSource


Requester
Local DB Subsystem
LocalDataSource
RemoteDataSource
Network Subsystem
Remote DB Subsystem
1: 'read'(subset, datasource)
2:[datasource is local] 'read'(subset)
3: 'return'(subset or error)
4:[datasource is remote] 'read'(subset)
5: 'send'(subset request)
6: 'extract subset'(subset)
7: 'return'(subset or error)
8: 'send'(subset or error)
9: 'return'(subset or error)
10: 'return'(subset or error)

1. Requester calls read and gives the wanted subset and the datasource to read from
2. if datasource is 'local', the subset is requested from the Local Datasource (e.g. CD)
3. Local Datasource returns the data subset or an error if it is not available
4. if datasource is 'remote', Local DB Subsystem requests the subset from Remote Datasource
5. Remote Datasource sends the request to the parent server via Network subsystem
6. Remote DB Subsystem receives an 'extract subset' command via Network Subsystem (see ExtractSubset Use Case)
7. Remote DB Subsystem returns extracted subset or error
8. Remote Datasource receives subset or error via Network Subsystem
9. Remote DataSource returns subset or error
10. Local DB Subsystem returns subset or error

### 3.5.5 User Interface - Navigational Paths and Screen Mockups

The Database Subsystem does not have a User Interface, thus this section is not applicable.