Requirements Analysis Document

PAID Project

Informatik XII

WS 1998/1999

Technische Universität München

January 21, 1999

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:

Clients, Developers

PAID Members:

Project Management:	Bernd Brügge, 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 Schrägle, Tobias Weishäupl, Alexander Zeilner
Database Team:	Osman Durrani, John Feist, Florian Klaschka, Johannes Schmid, Florian Schönherr, Ender Tortop
Learning Team:	Burkhard Fischer, Jürgen Knauth, Andreas Löhr, Marcus Tönnis, Martin Uhl, Bernhard Zaun
Network & Event Service Team:	Henning Burdack, Jörg Dolak, Johannes Gramsch, Fabian Loschek, Dietmar Matzke, Christian Sandor
STAR NETWORK 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, Götz Bock, Michael Winter, Sameer Hafez

Revision History:

- Version R0.1 9/15/98 Robin Loh. Created
- Version R0.2 12/16/98 Martin Uhl. Modified for TUM
- Version R0.3 12/21/98 Martin Uhl. Added automatical paragraph numbering
- Version R1.0 01/11/99 Marko Werner
- Version R1.1 01/20/99 Henning Burdack. Team RADs integrated

Table of Contents

1 General Goals

The general goal of the PAID system is to provide a powerful, extensible architecture for fast and efficient information exchange. This platform for active information dissemination should be fast and stable to meet its main requirement: intelligent worldwide distribution of most used information, i.e. DaimlerChrysler service, parts and vehicle data, provided by a common architecture. Data access should be transparent while the information itself is always up−to−date. The system should be scalable to accommodate a network of 6000 clients internationally. PAID should also function on several types of computing devices, such as desktop computers, laptop computers, and personal digital assistants (PDAs). The system should have the capability to take advantage of progresses in computer hardware as well as improvements in network technology and bandwidth. The architecture of PAID should allow for the addition of new data and functionality without the need for a complete system redesign. To provide this extensibility, the PAID system is subdivided into the following subsystems, with their own individual goals:

1.1 Authentication and Security

The PAID Authentication and Security subsystem has to ensure secure access to client and DaimlerChrysler parts information across all viable media in the PAID system. A PAID−wide user model is to be provided by the Authentication and Security Team. Within this user model, access rights are assigned to all users and/or user groups; besides, unauthorized access has to be prevented. Critical data is to be encrypted when sent over insecure or not trustworthy mains.

1.2 Database

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.

1.3 Learning and Adaption

The subsystem monitors the network and database activity and stores information about these activities. The subsystem then analyzes the data to expedite the decision making process of the PAID system. First, it will deal with a large number of clients (over 6000). Second, it will utilize machine learning and other intelligent algorithms in its data analyzation process. Third, it will investigate existing monitoring and diagnosis tools. Finally, it will provide reports of its analyzation with a graphical user interface.

1.4 Network and Event Services

The goals of the network subsystem are to reliably transmit messages and data between Daimler−Benz after−sales database servers and dealer machines running the PAID system. The information should be accurate and delivered in a timely manner. The communication between PAID−subsystems will also be provided.

All updates to the database should be pushed to dealers who are using the information using a multi−cast distribution policy.

1.5 User Interface and STAR NETWORK Integration

The User Interface and STAR NETWORK Integration subsystem provides user access to the PAID functionality and ensures a proper co−operation of PAID and STAR NETWORK.
The interface that the user will see is to be fully integrated into the STAR NETWORK interface. One of the main goals is to keep the interface of PAID really small, so that the user can focus his tasks on the data itself instead of getting the data.

From the point of view of the user interface the current system is the actual user interface of the STAR NETWORK client. It represents the frontend to the whole STAR NETWORK functionality, which could be for an end user from interest. The user has the possibility to access the different parts catalogues, after−sales information, etc., but at the moment he does not have any influence on the manner like the requested data being gotten and held up to date. To remove this last restriction, the PAID subsystem was designed.

While currently the STAR NETWORK client sends his requests to the STAR NETWORK server, which accesses his database directly and tries to respond to the requests, in the User Interface and STAR NETWORK Integration system model the STAR NETWORK server will not have his own database. The STAR NETWORK server will be modified so that all database requests will be forwarded to a PAID server. The answer will be passed back to the STAR NETWORK server, which then transfers the data to the STAR NETWORK client.

In fact the STAR NETWORK integration consists of helping develop and providing a translation module for communication between the STAR NETWORK server and the PAID server. The main goal concerning this aspect of the subsystem is to make as little changes to STAR NETWORK as possible, to allow for the further development of STAR NETWORK.

To sum it up it can be said, the current user interface of the STAR NETWORK client has to be extended by the components needed by the PAID functionality.

2 Current System

Within Daimler−Benz, aftersales information is created and distributed by different departments. The major information sources today are service, parts, and vehicle information. Depending on the type of information, the company utilizes a variety of different distribution channels. Table 1 shows the main aftersales information types, the respective end user applications and distribution channels.

These distribution channels are typically very reliable but also very slow and inefficient. For instance, the distribution of service, parts, and vehicle information to the worldwide Mercedes−

Benz sales organization is done monthly via a set of 12 CD−ROMs. This information is already partially outdated when it gets to the dealer. Looking at the information distribution channels that are available today through network technology, we can see that the current method of information distribution is too slow and inflexible to meet the demanding business requirements of Daimler−Benz.

Information Type	Application	Distribution Channel
Service Information	WIS	CD−ROM, Paper, Microfiche
Diagnosis Information	STAR DIAGNOSIS	CD−ROM, Paper, Microfiche
Parts Information	EPC	CD−ROM, Paper, Microfiche
Vehicle Information	FDOK	CD−ROM
On−line Car Configuration Data	MBKS	CD−ROM
Work Units and Operation Texts	ASRA	CD−ROM
Damage Codes	VEGA	On−line

Table 1. Main aftersales information types, applications, and distribution channels

Integration of different information types is also an important issue. As shown in Table 1, there are currently seven disparate information applications that dealers must consult. Additionally, many dealers also have their own internal information system that stores data of interest to the dealer, such as customer information and inventory. Because data for the Daimler−Benz information systems are only updated monthly, important interim bulletins and updates are often distributed via paper, forcing the dealer to consult multiple sources for the most recent information. The variety of systems currently used also results in instances of double or triple data entry.

Daimler−Benz is constantly extending its business in terms of new product lines and new models of existing product lines. The amount of aftersales information is increasing due to the introduction of these new products. With the introduction of new aftersales information systems, additional information distribution channels are created, which will lead to a proliferation of distribution channels. This makes the information management process (creation, publishing, distribution, installation), from a technology and management point of view, very complex and expensive.

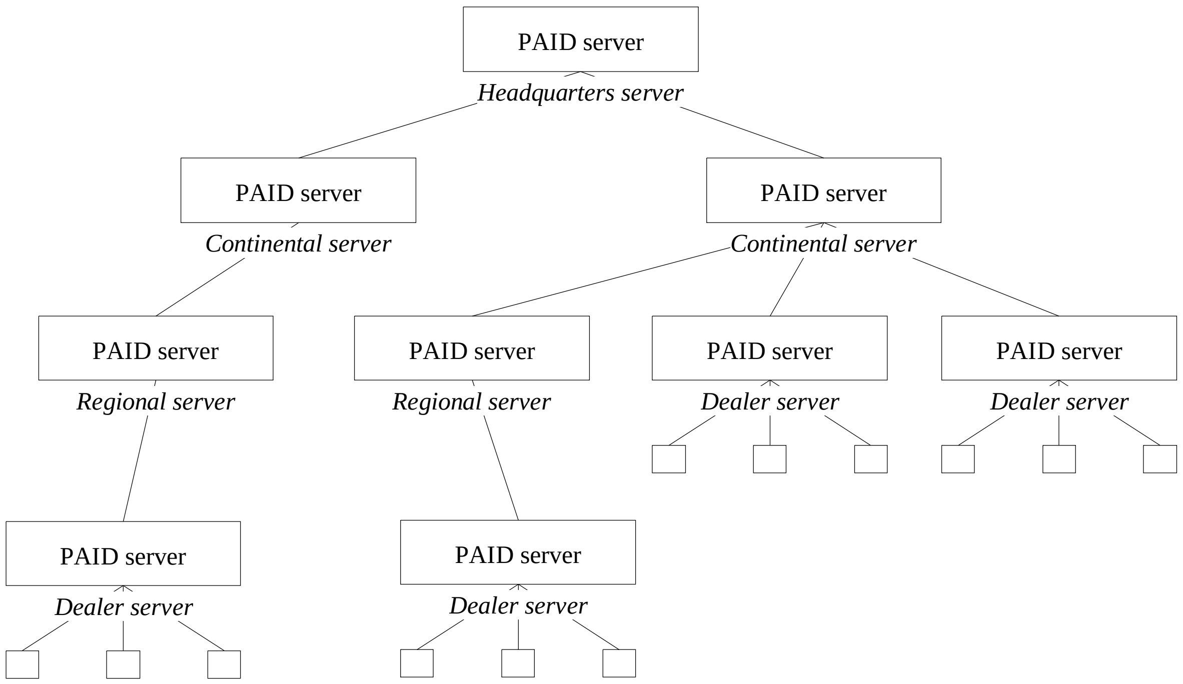
In summary, a reliable, flexible, and scalable solution for active information distribution is needed. In the next section, we will present our proposal for a new information distribution architecture that will not only improve current information distribution processes, but that will also be extensible and flexible enough to allow for the addition of new types of aftersales information.

3 Proposed System

System Model Overview: Overall System Model

The PAID system consists of an arbitrarily deep hierarchy of almost identical PAID servers. The top PAID server is located at DaimlerChrysler headquarters and is connected to the main STAR NETWORK database. For intelligent caching and replication of data, each PAID server has a local database which contains frequently accessed subsets of the main database. The bottom PAID servers are located at the dealers' and contain a modified version of the existing STAR NETWORK server. Existing STAR NETWORK clients will connect to this STAR NETWORK server.

The following figure shows an example hierarchy of PAID servers with four dealers on two continents. Each dealer in this example has three clients.

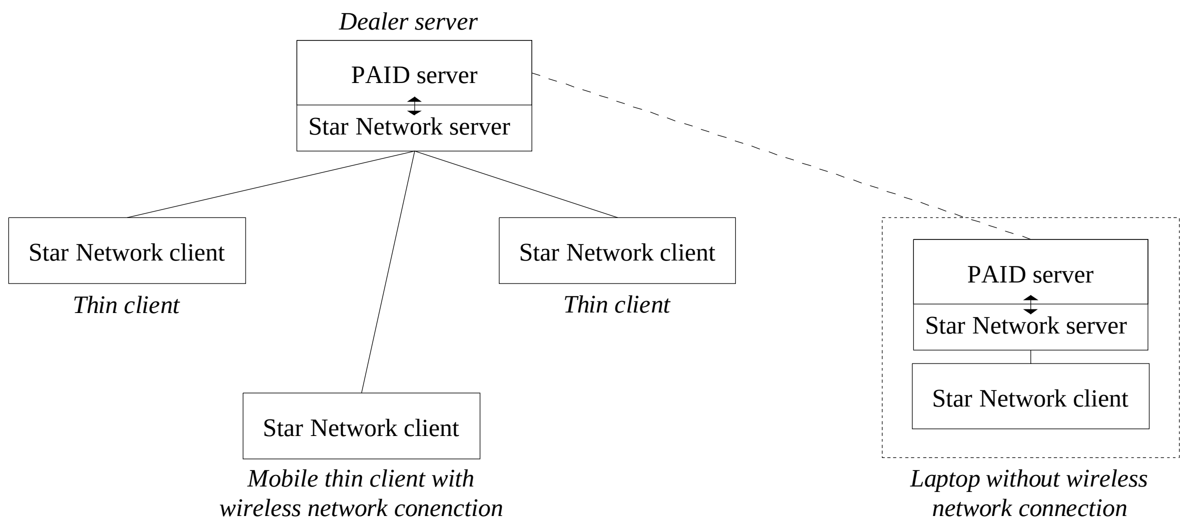


When a dealer requests information using his STAR NETWORK client, the request is sent via the STAR NETWORK server into the PAID system. If the dealer's own PAID server can answer the query, it does. If it cannot, it forwards the request one step upwards in the hierarchy, for example to a regional PAID server. Now this regional PAID server tries to answer the query, and so on. For improved fault tolerance, a PAID server can also be configured to send requests to an alternate PAID server if its own parent server is not available.

When a PAID server notices that a certain subset of the database is being accessed frequently, and it does not have this subset in its own local database, it subscribes to this subset from its parent server. This way, frequently required data migrates downwards towards the dealers requesting it, speeding up requests. For an additional performance gain, answers to queries get cached for a short time in case the same query is asked twice in a row.

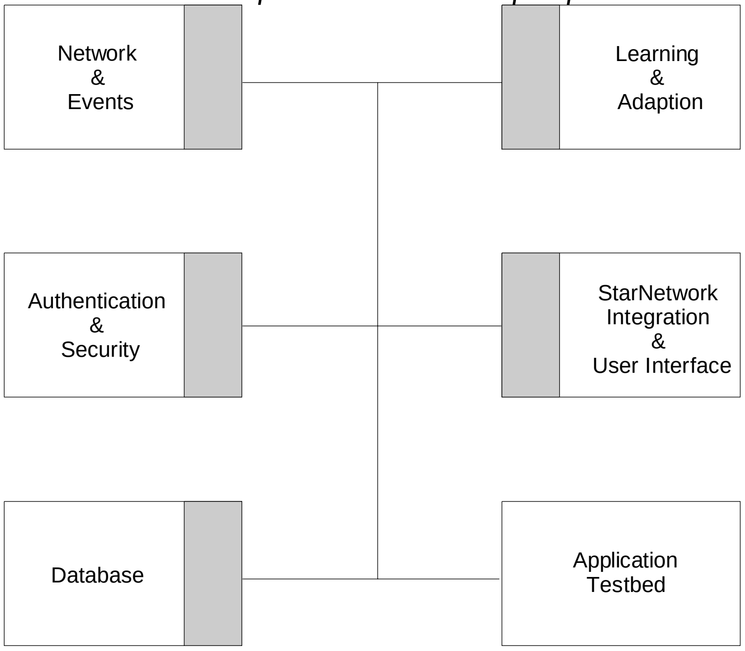
The top PAID server feeds updates to the main database into the PAID system and forwards them downwards to its child servers which have subscribed to the relevant data. The updates migrate down to all the PAID servers which need them. This ensures that all queries are answered with up−to−date data.

The following figure shows the configuration of clients and servers at a dealer. Since the existing STAR NETWORK clients only work when they have network access to a STAR NETWORK server, a mobile client must either have wireless network access or locally installed STAR NETWORK and PAID server software.



System Model Overview: Internal System Model

The PAID server software is divided in five functional units:


Network & Events
Learning & Adaption
Authentication & Security
StarNetwork Integration & User Interface
Database
Application Testbed

- **Network & Event Service**
  It enables the network connection to an upper PAID server or to a lower server. Incoming requests are received and given to *Authentication & Security* for decryption and authentication issues. It is responsible for building an event object that can be used within the event service.

- **Authentication & Security**
  This module allows encryption and signing of messages before they are sent over the network. It can also check signatures from incoming messages and decrypt them. Furthermore it will receive the event object built by the *Network & Event Service* and check whether it contains a database query and if it is one, whether the sender of the request is allowed to view this kind of data subset or if he might have to pay for it (billing). It can also authenticate a user's login on a *STAR NETWORK* user interface.

- **Database**
  The database module enqueues all queries it receives and tries to answer them. If a requested information is not part of the local database, a new request to an upper server will be issued while the enqueued query waits. The module has the ability to subscribe to a whole data subset on an upper server and to receive these subsets and updates. The database can also filled by CD−ROM.

- **STAR NETWORK / UserInterface**
  It has to ensure that the query asked by a user is transformed to standardized query format, and, if it is necessary, to send it to the next upper server. If the PAID server resides on the same machine, it can directly issue a query event on the *Event System* and bypass authentication and encryption/decryption for better performance. It has to provide integration into the existing *STAR NETWORK* system and particular user dialogs to meet the needs for all user interaction.

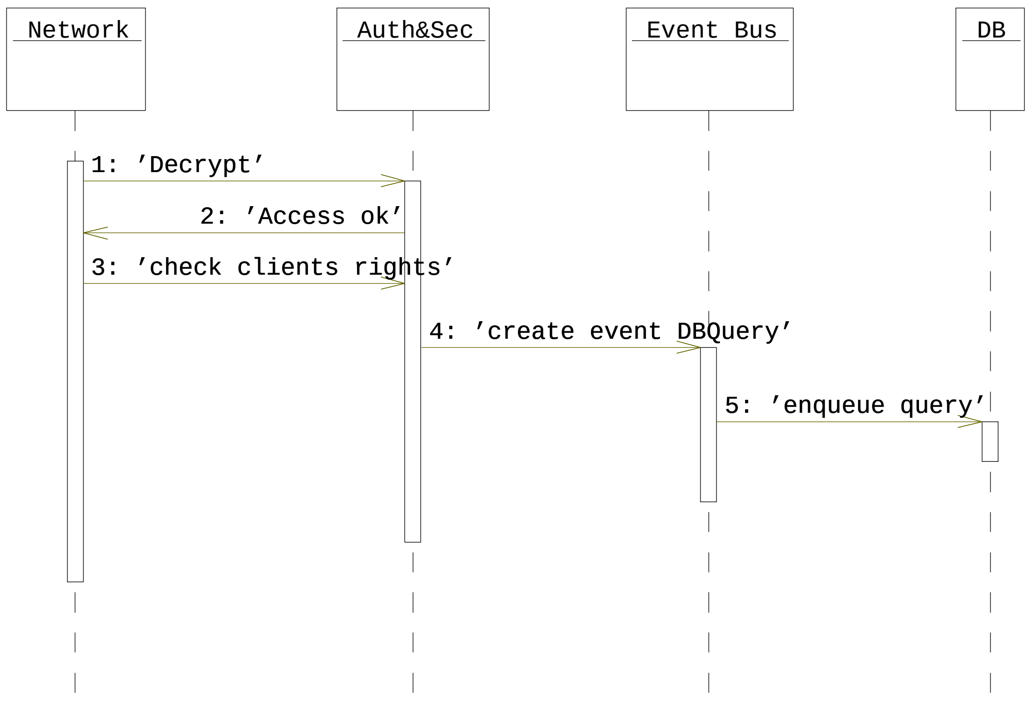
- **Learning & Adaption**
  If several incoming requests belong a particular subset, it might be sensible to fetch the whole subset and subscribe to regular updates. The task to determine when this should take place belongs to the *Learning & Adaption* module. It directly notifies the *Database* module to subscribe a subset. It uses special learning algorithms for that and logs all incoming database queries.

For testing purposes, another functional unit will be created. It is not necessarily a part of the final running PAID server:

- **Application Testbed**
  It will be used to check the contents of the events on the *Event Service*. Thus it can subscribe to any events of interest. It may also use special methods provided by the modules to explore internal states of them.

The communication between the units is based on the event service provided by the Network & Event Service and a supporting middleware based on CORBA. Only the communication between the Network unit and the Authentication unit relies on functions calls so that unauthenticated messages from the network do not occupy the event bus mechanism.

An example for this might be the following scenario. It describes the actions when an incoming network message arrives. This message contains a database query in the example.



The Network unit has received the message and calls a Authentication module function for decryption and authentication of this message. If the message appears to be all right, the Network unit creates an event for the database query and passes this event to the Authentication module again. There is checked whether the client has the right to view this particular data subset or if it perhaps has to pay for the answer. When access is granted the Authentication module issues the query event on the event bus system. The Database module has subscribed to those query events and receives our query. Then, the event is enqueued and processed. The answer is carried back on the same way.

3.1 Authentication and Security

3.1.1 Overview

The Authentication subsystem will provide secure access to the PAID system in terms of authentication and authorization, verification of all queries, encryption of crucial data, and smart user profile handling. The user model is in general of hierarchical order, according to the network architecture. Nevertheless we distinguish all accounts between *Administrators*, *Computers* (i.e. servers) and *Users.*

Computers can log in and out and query requests; they are authenticated by password only using a machine account on the upper next level PAID server. We propose that the user needs more sophisticated authentication, e.g. by a Java card which contains a much longer password and/or key and a PIN. The *User* can view their profile on screen and query requests.

The *Administrator* will have a profile but will not be able to gain access to any DaimlerChrysler data, be it with a Java card and PIN or a password. He will only be able to control user access, modify user profiles and add and delete accounts.

Dealer access rights can only be given by the higher instance (i.e. a DaimlerChrysler administrator). Each dealer administrator gets a certain upper limit of access rights that he can assign, depending on the dealer’s relationship to DaimlerChrysler, to other users at the dealership. The administrator is personally responsible for which access rights he gives to e.g. a mechanic.

The ’normal’ users *Computer* and *User* can be grouped by some criterion. Such a *Group* can consist of none, one or more *Users* or *Computers* or *Groups*, but at least of two of these.

In accordance with the hierarchical and thus potentially unlimitedly deep network there is no difference between local and remote data requests, so that every authorization will require authentication by one means per system, that is, for instance, a Java card combined with a PIN for every human user whereas machine accounts are authenticated by password. Authentication will be handled throughout the different network and server levels. The authentication tool − the *Device for Authentication and Verification* DAVE − for the Users should not be fixed or dictated but be able to be changed, so the interface to it should be adaptable to other security systems than Java cards (i.e. downgrading of the security level as well as upgrading should be principally possible for long term modifications on the PAID system).

Data which is transported via certain means has to be encrypted whereas all data in stored databases is not encrypted by the Authentication subsystem. However, data on CD or DVD databases is encrypted.

Overall, the Authentication subsystem is embedded in the PAID system as a provider for mainly three tasks: encryption, authentication/authorization and ensuring a secure transmission.

3.1.2 Functional Requirements

The functional requirements of the Authentication subsystem are to provide secure access to the PAID system and all the data contained within, locally as well as remotely.

To receive access to local or remote data, a user will be required to authenticate themselves using a Java card and PIN. This card will contain the user's authentication information, which will be used to log the user into the database. Whether or not a user has access to the information they have requested will be determined by the user model provided by the Authentication and Security Team.

All secret or important data or messages are encrypted and signed when transmitted.

Except for a login request all this functionality is transparent to the user.

3.1.3 Nonfunctional Requirements

3.1.3.1 User Interface and Human Factors

For the authorized user (who is a human being), the only interface to the authentication system will be during login. At that time the user will be asked to identify themselves by inserting their Java card into the attached reader and type in their PIN to open a new session. For all other interactions the Authentication subsystem will be encapsulated and fully transparent so the user will not see it. Every query will be verified to prevent unauthorized access to information. After a certain period of idle time, the user session will be ended. This amount of time will be defined by the administrator belonging to the specific user and held to certain limits.

The Java card is being implemented as a preferred security mechanism because it can provide a higher level of secure transactions than a regular user/password combination. The card allows for the storage of a large key, which will act as a password. The larger key increases the level of difficulty and time required to break into the system. By requiring the user to have the card when opening a session in which they could send requests for any data, we are providing increased control over access to data stored in DaimlerChrysler's databases. The PIN of every user increases security to the system as it prevents someone who found a Java card from using it.

3.1.3.2 Documentation

Documentation describing the user model and security rights will be provided to the administrators of the PAID system. In addition, the administrators will receive a list of possible security limitations known to the developers as well as the methods employed to prevent complete loss of security in case of a successful hostile attack. Also, they must be given information about their responsibility when assigning rights. A description of login, logout, and query procedures in addition to a manual on the use of the Java card will be provided to all users. PAID software documentation must be provided in Javadoc. All project documents will be published in StarOffice formats or HTML.

3.1.3.3 Hardware Consideration

The system must be capable of running a Java virtual machine. In addition to the desktop requirements for running the PAID system, a DAVE compatible with the PAID system is required on every system PAID will be run on.

Outside of the actual machines the PAID project will be implemented on, DAVE (e.g. a Java card reader if it is decided to use Java cards as authentication means) is the only hardware that PAID is directly concerned with. The quality and reliability of this hardware (and the cards, in case) will directly affect the progress of this project.

As the interface of the authentication tool will be made adaptable to any other implementation, in future also biometric systems like fingerprint or iris recognition, or any other authentication tool may be used.

3.1.3.4 Performance Characteristics

Login and logout should not take more than some seconds, and authentication and authorization tests may only be required at the opening of a user session. Remote connections to the PAID system should also be handled quickly (assuming there is no noticeable lag in the network connection) because only the servers have to announce themselves on every hierarchy level. Nevertheless, every request has to be authenticated, moreover on every level on its way up to the HQ server.

When transmitting data across public and private networks, encryption will be employed to protect the data from theft. It will require time to encrypt on the server side and decrypt on the client side. Also in the dealer network and maybe even on the user’s computer transmitting of data requires encryption, as stored data itself is not encrypted, and the possibility of eavesdropping or theft at the dealer’s site may not be excluded.

3.1.3.5 Error Handling and Extreme Conditions

We distinguish between several extreme conditions and severe errors:

- Unauthorized access (try) to crucial data by any user. If someone or some server requests information to which they do not have access rights according to their profile, they should be notified, after some tries warned and finally their session should be quitted. Higher instances should be informed and a trace of the user’s acting should be logged.

- Access try with a corrupted key or other identification means. First, the access will be denied until a correct key is provided. If one is not provided at 3 successive tries, the administrator is notified that someone is using an invalid key and on which computer the access was tried. If the login or request was tried to be made remotely, the administrator is informed, too, and the IP address from where the request originated is noted.

- The session is idle for some amount of time, as specified in the user’s profile. For security reasons, the session is terminated in this case so that a new login is necessary for any requests.

- Unexpected shutdown. In the case of an unexpected shutdown of the program, all sessions have to be terminated.

If a Java card is used for authentication:

- The Java card is lost or destroyed or the PIN was forgotten. Login and access is not possible then. Users will be required to contact DaimlerChrysler (or their administrator) to obtain a new card that will contain a new key. If the user just needs a new PIN, she has to ask her administrator. The user is responsible to inform their administrator of the loss, and the administrator has to delete all data from the user profile that is relative to the lost card.

- The card reader does not work. If the reader is out of order, this error must be recognized and the user will be informed to get things fixed by service personnel (if the distribution of the readers is done by DaimlerChrysler, they could contact e.g. a DaimlerChrysler service hotline). The user must be informed if the card is not properly inserted or if it is destroyed (see above).

3.1.3.6 System Interfacing

There are several actions or proceedings initiated by the users (*Users*, *Administrators* and *Computers*) which use the Authentication subsystem and require interfacing with other (sub)systems. This interaction is needed when there are inputs from outside the authentication systems or outputs to another system. As our system is usually transparent to the user, almost every user action is passed to the authentication system by another system; also, no security relevant resultˆ is directly transmitted to the user.

Thus, the Authentication subsystem is either to be seen as a in−between unit through which all messages have to be passed, or as a separate tool which has to be consulted by the other subsystems if authentication, authorization and other security issues are required. The system that handles the messages and data flows among the subsystems is the Event Service.

All actions and inputs done by the *User* are transmitted via the User Interface subsystem. We consider them to be located directly adjacent to the STAR NETWORK system.

All messages and requests from outside a PAID server are sent to the Network subsystem. The Network subsystem passes all those transmissions first to the Authentication subsystem which has to decide whether the received data packet can be trusted. If so, encrypted data has to be decrypted. All data is passed to the Event Service. Following actions begin or end with a command sent by the Event Service. If such a command is relevant for Authentication subsystem, i.e. something has to be en−/decrypted or authentication and authorization has to be done, Event Service informs the Authentication subsystem.

Therefore, the only direct interfaces of the Authentication subsystem are to the Event Service and Network, whereas logically there is interaction with all PAID subsystems.

Considering this, the User actions require interfacing as follows:

At login, the User Interface subsystem notifies the Authentication subsystem (via Event Service), which in turn verifies the user as follows: The card reader is initiated and the User Interface Subsystem has to be notified (via Event Service) to ask the user to insert their Java card. When the User Interface subsystem signalized that the card is inserted, the card data is read and the User Interface must prompt the User to type in his or her PIN. After these inputs are done the user is verified and security, authentication and authorization tests are performed. To do that, the Database subsystem has to be asked for the user profile data (via Event Service, that generates an appropriate command). Finally, the Authentication subsystems opens a session and notifies the Event Service of the successful login.

At logout, the User Interface must signalize the Authentication subsystem that the user wants to log out. All other subsystems have to be notified by the Event Service of the user’s request and asked to close all current user sessions and quit activities. Only when all subsystems have reported that their particular logout routine is finished, the User Interface can be notified to tell the user that the logout has been successful.

If the User queries a request (vehicle parts data or a request to view their profile), the query is forwarded to the Authentication subsystem by the User Interface. With help of the user profile security and access checks are performed.
If these fail, the User Interface system has to be notified to inform the user. After three unsuccessful tries, the User Interface has to be told to quit the program, while the Authentication subsystem has to inform the Administrator.
If, otherwise, the query turns out to be valid, on the one hand the Learning subsystem has to be informed of the new information request. On the other hand, the Database subsystem has to be contacted to find out if the data is stored locally. If not, the network subsystem must be told to forward the query to higher levels and databases (of course, after the message was encrypted by the authentication system). The *Computers* (servers) on the way up have to be authenticated and authorized on the next higher level; the only difference now is that the query does not come from the User Interface but the network subsystem. In any case, i.e. on any level, the Learning subsystem also has to be notified. As soon as a database contains the requested information, the Authentication subsystem on that level encrypts the answer and passes it to the network subsystem to get the data transmitted back to the requesting user computer.
If the information resided locally (what is always the case if the User wants to view their profile), the Database subsystem has to handle display, if the data arrived remotely, it has to be decrypted and at first cached, and the Authentication system has to notify the User Interface system to display the requested data. Caching the data can be avoided if the learning system decides the data to be stored: in that case, the data has to be transmitted to the Database system.

The second user type, the *Administrators*, requires similar interfacing for their actions. In general, the Administrators just alter user profiles. This action is an access to the database and does not differ (as to interfaces) from the request for viewing the user profile (but now the Administrator is also allowed to write).

The *Computers* (servers) again show the same interface structure. They represent as a ’virtual’ user the PAID server. The *Computers* act like *Users* but do not need the User Interface and only have a machine password. Therefore, the cases login/logout and querying a request (most likely requests for DaimlerChrysler data) also apply to the *Computers*. They notify the upper next Network subsystem directly; the Authentication subsystem there tests if the connection is trustworthy, and the authentication is being done by just checking the machine password. As the only difference to the *User*, the *Computer* is notified of results of his query directly, also if authentication fails or errors occur.

3.1.3.7 Quality Issues

The Authentication subsystem will ensure that no data is given to unauthorized parties. Measures will be put into place to prevent a successful hostile attack on the PAID system from being too damaging.

The Authentication subsystem will reliably handle all user requests for important DaimlerChrysler data. Misuse of the system is prevented. If remote connections are needed, the Authentication subsystem will test their trustworthiness and also authenticate the remote computer. Besides, all data and messages will be encrypted. As the system acts in between the other subsystems, absolute reliability and correct function at all times is a must.

As with the rest of the PAID system, authentication should be platform independent. At no time will PAID be operational if authentication is not functioning.

3.1.3.8 System Modifications

Future enhancements and modifications to the Authentication subsystem can and will be all kinds of technological progress. This may be improved encryption techniques for data transmitting, and new ways and technologies for increased authentication security. The Authentication subsystem will be designed to cope with new developments. Modular architecture will provide adaptable interfaces to changed encryption routines as well as to complete new authentication means.

Thus, authentication by Java card with DAVE being a card reader can be replaced by less strong user name/password schemes if it is the client’s favorite, but also easily be upgraded to new systems like iris scan or voice or fingerprint recognition.

Another modification can apply to big dealers if authentication by Java card is preferred: To save DaimlerChrysler personnel, large dealers may (have to) buy a Java card writer to make their cards. In this case, the Java card has to be written from an existing profile in contrary to the ’normal’ case where the profile is initiated by the card. Additional methods and use cases would be necessary.

3.1.3.9 Physical Environment

The physical environment the Authentication subsystem is concerned with is highly dependent on the authentication tool that is, primarily, the Java card and the card reader. No session can be opened and no authentication will work if either the card reader is inoperable, the card is damaged or the card reader cannot read the security information. Also, care must be taken when dealing with the Java cards, since they are sensitive to the environment. The reader must be able to work in rough environments (on the street, operated by a mechanic) and must be designed to be handled by inexperienced computer users; also, error messages and other instructions have to be aimed at these clientele.

3.1.3.10 Security Issues

Security is the heading all procedures in the Authentication subsystem should keep to. Thus, every aspect (including those mentioned above) is also a security issue. Nevertheless, the user model and its handling in all occurring conditions is the core of the authentication and security unit.

So only administrative users will be able to change user profiles or user access rights. Every data access will be verified to ensure that a user has the right to access the data requested.

In building the security model, certain specific types of attacks are being considered. Authentication is concerned with protecting the system from persons pretending to be dealers, persons (even at the dealership) who want to exceed their access rights and from people attempting to listen to network communications in order to capture data transmitted from some server to some client. In addition, we are concerned with unauthorized users attempting to gain access by imitating the Java commands internal to the PAID system. Finally, while the physical dealer machines cannot be protected by any software methods, measures will be in place to make accessing the data stored locally as difficult as possible.

3.1.3.11 Resource Issues

Administrator users are responsible for handling user accounts, adding, deleting and modifying etc... Also, the main system will be backed up by the administrators on the specific (network) level. While the installation of the core/server PAID system including the databases will have to be done by the administrators, the installation of the local part of the PAID systems will be the responsibility of the end user. Installation and maintenance of the main PAID system and its corresponding databases will be done by the PAID administrators.

3.1.4 Constraints

The entire system must be written in 100% pure Java. Development will be done using the Together/J CASE tool. If the Java card is chosen as an authentication tool, we will use the Schlumberger Java card. The prices of card readers are constantly dropping and are under $ 500 now.

In any case, any DAVE must be accessible in Java or offer a Java card API.

Furthermore, the user model as the base for authentication and authorization is highly dependable from the network layout. But above all: overall security is just as good as the security of the databases.

3.1.5 System Model

For a better understanding of the following text, we give an overview of the PAID server hierarchy. PAID servers can connect to "parent" PAID servers. As we see the network structure as a tree (i.e. hierarchical) we call the tree level of the parent server "upper level". From a "parent" point of of view there are connections from a "lower level".

3.1.5.1 Scenarios

Note: For all scenarios, we assume the Java card to be the authentication tool.

PAID Installation Procedure

George is a new DaimlerChrysler resp. Mercedes−Benz affiliated dealer. Since he wants his dealership to take part in the PAID system, he sends his PAID registration request and information to his DaimlerChrysler administration center. In this form, George additionally lists his dealership staff that may also get access to the PAID world, but he has to verify their employments. Another information George has to deliver is who will be the administrator at his dealership. This person will be responsible for the user model, accounts and access regulation at the dealer’s site and has to be named exactly. George’s administration center is well−known to him and may be the national dealer server center (COMETˆ)or, e.g. for the USA, MBNA ˘−maybe the administration tasks there can be passed down to state−wide sub−centers.

In any case, Jane, an administrator at such a center, receives George’s request and creates an account and a user profile for George’s dealership. Maybe she has to ask DaimlerChrysler Headquarters what rights George can be given, or his dealership can be categorized into a special dealership Group with applicable rights. The specific access rights that Jacob gets depend on location of the dealership and his level of affiliation with Mercedes−Benz. This account for George’s server also gets an entry which says for which parts of the databases at DaimlerChrysler George will be billed every time someone of his dealership staff clicks at that specified pieces of information.

Jane also prepares Java cards containing authentication keys and passwords for all people because George’s dealership is not that large that he wanted to buy a card writer. Jane does not have to register the individual Users, but the cards get invalid after two years.

Once she has completed this process she mails George and his personnel their Java cards, PAID manuals (administrator manual for George or whom he designates for administrator, end user manual for each employee). Also, George gets a login name and password for his server (the dealer machine account) saved in the dealer profile Jane created. The PAID and STAR NETWORK software itself can be delivered on CD’s or DVD’s or via the Internet or other electronic means.

At the dealership, George’s Administrator, Jack, starts the setup procedure. The software has a built−in predefined Administrator account initially with a blank password. This account does not have any rights to read DaimlerChrysler data. It is just used to set up the software, to create new user accounts and to specify the computer’s login name and password (this *Computer* login/password is used when the *Computer* − which is the dealer server − connects to the remote PAID server). In the course of the setup procedure Jack can change the administrator password to protect the *Administrator* account. Jack defines the *Computer*’s (dealer server’s respectively) login name and password − Jane had sent those data to him. George’s server saves its login name and password for further use.

Jack defines new *User* accounts for the staff and assigns this user accounts to user groups (e.g. Joe/mechanic, George/dealer). Jack also can predefine the *User* preferences (e.g. he knows that Joe will always need data of the A−class). Of course, these preferences will be overwritten by the learning algorithms. To finish off the *User* accounts, Jack has to pull all Java cards through the attached card reader. By this the cards are assigned to the Users. Jack has to note which card belongs to whom. The computer automatically generates PIN’s for the Users that will appear to them (only) the first time they log in with their new card. Jack also has to create machine accounts for the two dealer laptops. He sets up special profiles for which he does not need to provide Java cards. The PAID Administrator documentation papers help him do this.

Now, Jack has the software files installed, user accounts have been created completely. But there is no DaimlerChrysler data in the database yet. Jack has two possibilities now:

1. Database installation from CDROMs that Jane has sent him. As mentioned above the database on the CDROMs is secured. Jack will need to have a login name and password to access the database. But he is an *Administrator* and cannot access DaimlerChrysler data. So he has to connect to the remote PAID server. For this connection the *Computer* login name and password will be used. If the login succeeds a name and password will be transferred to the dealer server and the database on CDROM can be opened. Now Jack can copy the data he initially needs from the CDROMs to his local database. After copying the initial data the normal update process over the net starts.

2. Database installation over the net. Jack connects to the remote PAID server. If the login succeeds the dealer server requests an initial database update (in contrast to the normal incremental database updates). After copying the initial data the normal update process over the net starts.

Now PAID is ready for normal use.

Login

It is a sunny day in Downtown Phoenix, Arizona. Barbara, a sales executive at John's new Mercedes−Benz dealership, switches her workstation on. Immediately, the STAR NETWORK main desktop window pops up. After she clicked on the Enter button, the computer asks her to insert her Java card. Barbara has to fumble about in her purse and can finally insert the card. The card reader light goes on. On the server back in the building, Barbara's profile is read and checked with the card. In the bureau on the screen appears a message greeting Barbara personally and asking her to type in her PIN. After she has done this, there is another short message (Sessionfor Barbara opened successfullyˆ) and windows pop up − exactly those Barbara saw last night when she left, because she had decided to have it that way when the computer asked her some time ago. Now Barbara can put her card away again.

Accessing Information: from the Dealer Server and remotely from Other Servers

Barbara sits in front of her workstation at John's Mercedes−Benz dealership. The STAR NETWORK Software just wished her a nice day as she logged on. Barbara wants to review some M−class information which she frequently uses and therefore starts the appropriate STAR NETWORK viewer. Because it's used frequently the requested data is stored locally at the dealer server. At first the request will be authenticated. As Barbara is known to the system and has the right to query the database the request will be passed to the database subsystem which generates an answer. The answer will be transferred back to the STAR NETWORK Client and Barbara sees the data on her screen. If Barbara would be an employee of a non−affiliated dealer the system would have informed her about the costs of her requests. As the data was stored locally and no connection to a DaimlerChrysler server had to be established there wouldn't have been any costs so far.

A customer arrives and asks Barbara for the price of a door handle for his C−Class. Barbara doesn't know it because she never sold such a part. In fact she never looked at data from the C−Class. Barbara formulates a request to the PAID system. As above the connection to the PAID server is authenticated, Barbara's user rights are verified but this time the local database cannot answer the request. The PAID server establishes a connection to the next remote PAID server to get an answer for Barbara's request. Because John's dealer server is known to the DaimlerChrysler PAID server the connection will be allowed. The request arrives as an encrypted byte stream at the DaimlerChrysler PAID server. It is decrypted and access rights are verified. The remote PAID server queries its local database and passes an encrypted answer back to the dealer server. The answer will be decrypted and transferred to Barbara's workstation and finally Barbara sees the data on her screen. At a non−affiliated dealer the system would summarize the connection costs and inform Barbara about it.

Introduction of M−Class in Germany − Server Push Model

DaimlerChrysler introduced the M−Class. A new record set was created on the top level server. Now the administrator of this server has to select which direct clients have the permission to get the new records. He informs the selected clients about the new M−Class, so they could decide if they want to get the data regularly. If the selected clients are also servers, they have to select which of their direct clients are allowed to access the new records. This is done downwards (i.e. recursively) to the lowest client.

Session Time Out and Logout

Joe is a mechanic at Geoge’s dealership. He uses the computer that George had installed in the workshop bureau. Today he mustˆrepair an SL 600 ˘ its owner’s poodle has bitten the wooden covering of the automatic shift stick. Joe gets to the computer because he thinks he has to order the piece, and quickly finds the desired part. He discovers that they must have one of those stick covers left. With the exact parts number, he can find it in just a few seconds. Joe just wanted to start to renew the part when he sees it is lunch time. Coming back an hour later, he remembers he had forgotten to quit the parts software. But coming closer to the computer he sees that the screen has gone black. He presses a key and looks at the STAR NETWORK initial screen ˘ the system has logged him out and quit the session.

Barbara in Downtown Phoenix finishes her last tasks for today. She decides to quit the program and selects the menu point. A message appears that says that her session data will be saved now and that she please wait a few seconds for the program to terminate. Back in the server room, all data that Barbara did decide not to save and everything what is not needed persistently is deleted permanently. After this, Barbara can leave the workstation, and STAR NETWORK wishes her a good night.

Mobile Garage

A call comes into George s dealership requesting a repair on a car that has broken down somewhere far away. George sends his mechanic Bill, to figure out what is wrong with the car and try to fix it. Bill logs into a laptop, which was prepared by George to be used as a normal workstation or − like in this case − as a computer, which is independent from any servers above. In order to prepare a laptop like this, the Administrator of George’s dealership has to install a separate PAID−Server on the laptop and create a separate profile on the laptop, normally the same as on the dealerships Server (see "PAID Installation Procedure"). The laptop itself has a machine account on George’s server.

In order to prepare the laptop for the service outside the Garage, Bill has to log into the laptop (by using DAVE) finding out, if the relevant data is local or if he has to get it from an upper server. After saving all data locally, Bill can disconnect the laptop from the network and go to the broken car. Outside there he can use the laptop like at the garage (logging in by using DAVE), except for that he can only use local data.

Nevertheless, Bill will be able to get a connection to DaimlerChrysler for further information. In order to do this, he must connect his laptop to any PAID server in the PAID network and his laptop must be known to this server (the laptop must have an account on the server − see "PAID Installation Procedure").

Back at the garage, Bill can reconnect the laptop to the dealership−network and use it as a normal workstation.

Editing the User Profile, Adding/Deleting Users

Frank, the chief mechanic, decides to retire from the dealership, so Jerry, another mechanic, gets promoted to his position. The dealership informs his local administrator to requests more access rights for Jerry while requesting the cancellation of George’s account.

On receiving this request, Elaine the local administrator, modifies Jerry’s user profile and gives him more access. She also removes Frank’s account. After completing the process she asks Frank to return his Java card and she informs Jerry about his new access rights.

Some weeks later the dealership hires a new mechanic called Cramer to do Jerry’s old job. Again Elaine is informed to create a new user account. She uses an administrator program and a card writer to generate a new PAID Java card and registers it to the PAID System. After that she defines the new user rights for Cramer and hands out the card.

3.1.5.2 Use Case Models

Actors

Actors constitute everything that is external to the system and that communicates and interacts with the system, i.e. human users, external hardware and other systems. We consider the Authentication subsystem to be the (main) system of the following use cases and can find the following actors:

- **Client.** The user initiates every query and login and logout requests. Nevertheless, it is actually the User Interface subsystem that forwards all user requests and other actions of the user which are intended for the Authentication subsystem. Therefore, the Authentication subsystem only sees a PAID client.

- **Server.** This actor can be e.g. a dealer server which connects to the upper next server. The STAR NETWORK server is a Server to the PAID system.

- **Administrator.** This is the person at the dealership or at any other PAID network level who has to manage accounts and profiles of the users or computers (dealer servers) of the level below. An administrator has an own account without Java card, but does not have rights to read DaimlerChrysler data.

- **Timer.** The Timer comes to action if a programmed session time out is reached.

- **EventService.** This Actor represents the unit which does the message service between the PAID subsystems. For the Authentication subsystem, it handles all interaction with the Network & Event Service, the Database and the Learning & Adaption subsystems.

- **Network.** This is the PAID subsystem that manages all network tasks. The Authentication subsystems communicates directly with the network subsystem and not via the EventService.

- **DAVE.** The Device for Authentication and Verification is the only external hardware communicating with the Authentication subsystem. It can be a Java card reader or another authentication tool.

Note: The Authentication subsystem is transparent to the user. Also the user does not interact directly with the Authentication subsystem.

Use Cases

ClientLogin Use Case

Participating actor: initiated by *Client*, communicates with *DAVE, Timer*.

Entry condition: The *Client* wants to access the PAID system.

Flow of events:

1. The *Client* must be authenticated by the user inserting the Java card into *DAVE*, a card reader.

2. The data on the card is read and securely transmitted to the system.

3. As a result the authentication is performed (with the **Authenticate Use Case**).

4. The *Client* is granted access, the working screen is displayed, a session is started and the *Timer* activated.

Exit condition: The *Client* is logged in.

Exceptions:

- The *Client* sends wrong card identification data and/or wrong PIN more than three times. Then the *Client* is informed that it cannot get access. A log is written with any input data that could be get. The system is locked for further use (the Administrator can unlock it).

- The **Authentication Use Case** reports that authentication has failed. Then the *Client* is told that it does not have access. A log is written. If the *Client* tries again two more times, the system is locked.

Authenticate Use Case

Participating actor: communicates with *EventService.* (This use case is not initiated by a special actor but just used by other use cases.)

Entry condition: A user (User/Computer/Administrator) has to be authenticated.

Flow of events:

1. *EventSevice* is instructed to get the user profile.

2. The user is authenticated by checking the profile with the provided key and/or password.

3. The result is passed back to the use case that uses this use case.

Exit condition: The user is authenticated.

Exception: If the authentication fails, this is logged. The calling use cases have to further react on the negative authentication.

ServerLogin Use Case

Participating actors: initiated by *Server*, communicates with *Network*, *Timer*.

Entry condition: A remote connection is established and a PAID server (usually on a lower level) wants to login at a remote server (on the upper next level).

Flow of event:

1. The STAR NETWORK server or the PAID server of the lower level appear to the *Network* of the upper level as a *Server* that wants to login.

2. *Network* calls the **Authenticate Use Case** (on the upper level).

3. If authentication does not fail, a session for the *Server* is started and the *Timer* is activated.

Exit condition: The S*erver* is successfully logged onto the other server.

Exception: The authentication fails (in the **Authenticate Use Case**). In this case, the lower *Server* is told that it will not get access. All known data from the failed login is logged.

CloseSession [Time Out/Logout] Use Case

Participating actor: initiated by *Client*, *Server* or *Timer*, communicates with *EventService*.

Entry condition: *Timer* is up or *Client* or *Server* wants to quit.

Flow of events:

1. Case 1: The session *Timer* starts the logout procedure. It is displayed that *x* minutes are left until quitting. If there is no interaction, step 3 is initiated.

2. Case 2: The *Client* or *Server* indicates that it wants to quit and terminate the connection.

3. The session is closed. Further requests of this session are not possible. *EventService* is instructed to inform the other subsystems of the quitting. (and, for the *Client* only, the system screen returns to the login window).

Exit condition: The session is closed.

Exceptions: The network connection breaks down. The remote *Network* initiates Step 3.

AddUser Use Case

Participating actors: initiated by *Administrator*, communicates with *DAVE*, *EventService.*

Entry condition: The *Administrator* wants create a new user (User/Computer) profile.

Flow of events:

1. The *Administrator* can start a profile alteration. A form is presented for the required entries.

2. A Java card must be provided to complete the creation process. According to the hardware equipment he *Administrator* may be able to write Java cards on their own. In the case a writer is available, this write−enabled *DAVE* can be used to make a new card with a new key. Otherwise, a card (from DaimlerChrysler) must be used with a read−only *DAVE* to setup the profile.

3. The *EventService* is instructed to get the new profile saved when the *Administrator* has done his work left the editing screen.

Exit condition: One or more user profiles are created.

Special requirement: The user ID must be unique.

EditUser Use Case

Participating actors: initiated by *Administrator*, communicates with *DAVE*, *EventService.*

Entry condition: The *Administrator* wants edit a user (User/Computer) profile.

Flow of events:

1. The *Administrator* can start a profile alteration. A form is presented for the required editing entries.

2. If a new key or PIN must be created, a Java card has to be provided to complete the editing process. According to the hardware equipment the *Administrator* may be able to write Java cards on their own. In the case a writer is available, this write−enabled *DAVE* can be used to make a new card with a new key. Otherwise, a card (from DaimlerChrysler) must be used with a read−only *DAVE* to setup the profile.

3. The *EventService* is instructed to get the new profile saved when the *Administrator* has done his work left the editing screen.

Exit condition: One or more user profiles are edited.

Special requirement: The user ID must be unique.

DelUser Use Case

Participating actors: initiated by *Administrator*, communicates with *EventService.*

Entry condition: The *Administrator* wants to delete a user (User/Computer) profile.

Flow of events:

4. The *Administrator* can start a profile alteration.

5. The *Administrator* can select a Delete function and is asked if they are really sure to delete the profile.

6. The *EventService* is instructed to erase all profile data from the database.

Exit condition: One or more user profiles are deleted.

Encrypt Use Case

Participation actor: initiated by *Network*.

Entry condition: Some data is to be encrypted.

Flow of events:

1. The *EventService* gives a notification to encrypt the data.

2. The data is encrypted and passed back to *Network*.

Exit condition: The data is decrypted.

Decrypt Use Case

Participating actor: initiated by *Network*, communicates with *EventService*.

Entry condition: Incoming data is to be decrypted.

Flow of events:

1. The *Network* passes the forwards the encrypted incoming data.

2. The data is decrypted and passed back to *Network*.

Exit condition: The data is decrypted.

CheckPermissionOnCommand Use Case

Participating actor: initiated by *Network*, communicates with *EventService*.

Entry condition: A command (request, message, data stream,...) is to be authenticated.

Flow of events:

1. *Network* passes a command.

2. Permissions are checked with the command initiator profile and with help of identifiers of the command.

3. If the command is valid it is passed on to the *EventService*. In other cases the *EventService* is informed about an invalid command.

4. Exit condition: The permission on the command is checked and the result passed to *EventService*.

AuthenticatePacket Use Case

Participating actor: initiated by *Network*.

Entry condition: A packet that network received be checked on trustworthiness.

Flow of events:

1. *Network* contacts the Authentication system and passes the data packet.

2. The packet meta−data is checked if valid and to be trusted.

3. The result (be it positive or negative) is passed back to *Network*.

Exit condition: *Network* knows whether to dismiss the packet or not.

AuthenticateSubsystem Use Case

Participating actor: initiated by *EventService*.

Entry condition: When the software starts all subsystems have to register at the *EventService* what kind of events they can subscribe. The subsystems have to be authenticated and permissions have to be checked.

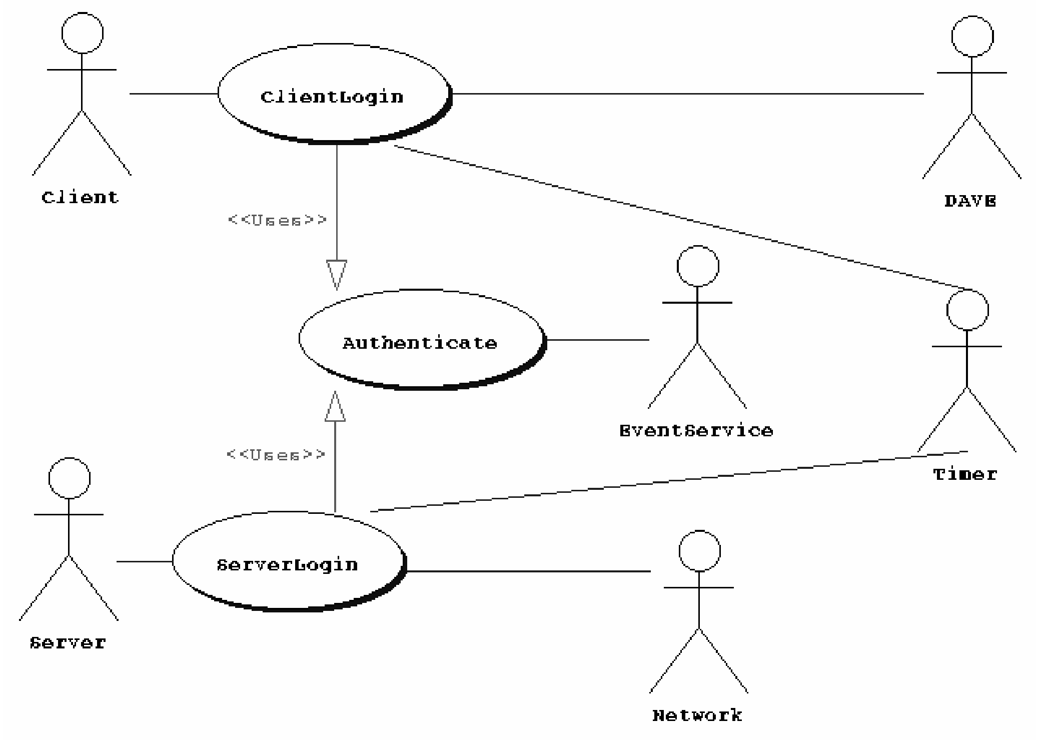
Flow of events:

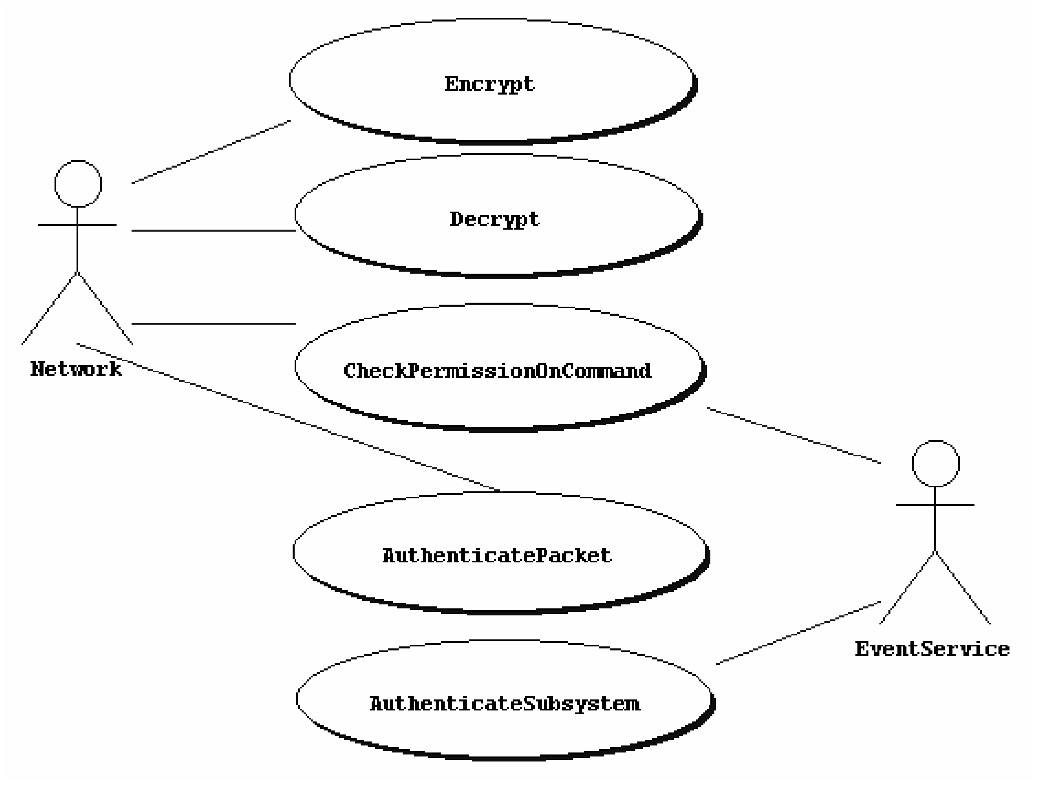
1. The *EventService* passes signatures of the subsystems that want to subscribe.

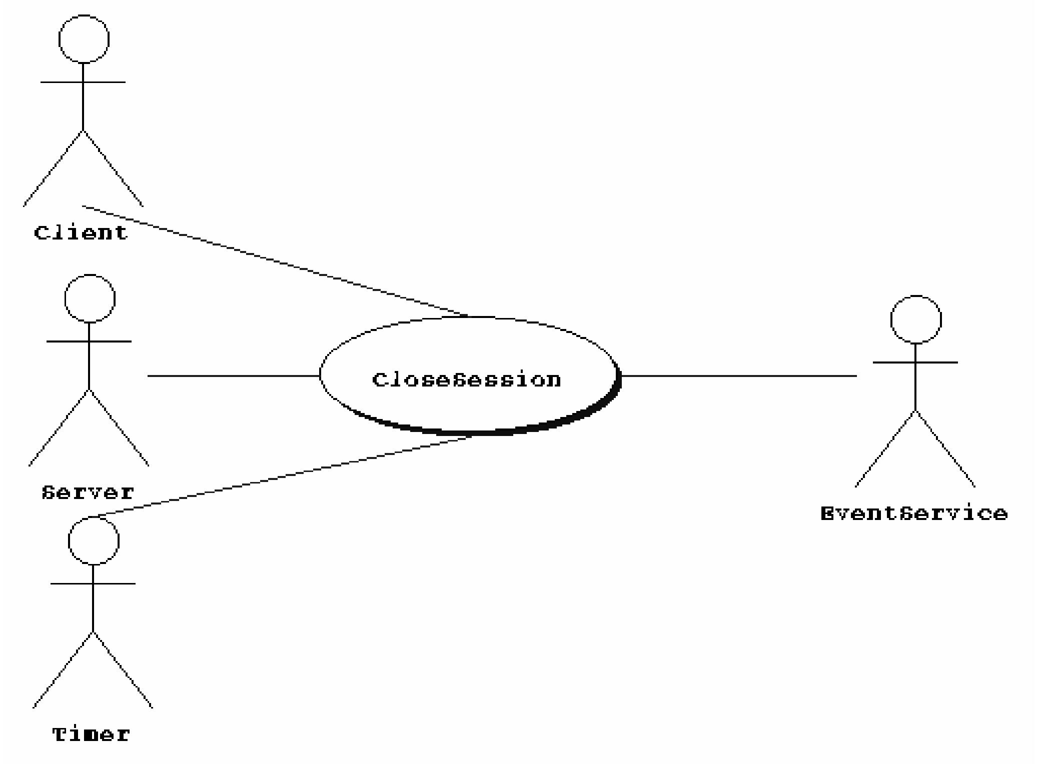
2. According to the signature it is decided if and what the subsystem is allowed to subscribe from the *EventService*. (this is valid until shutdown)

Exit condition: The subsystems are registered with the *EventService*.

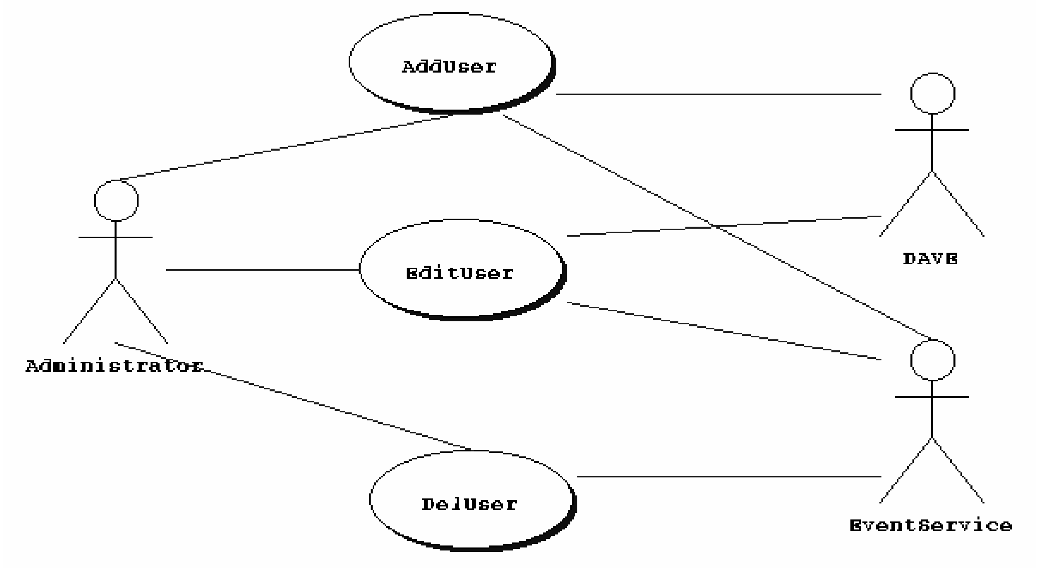
Client & Server Use Cases




Encrypt
Decrypt
CheckPermissionOnCommand
AuthenticatePacket
AuthenticateSubsystem
Network
EventService

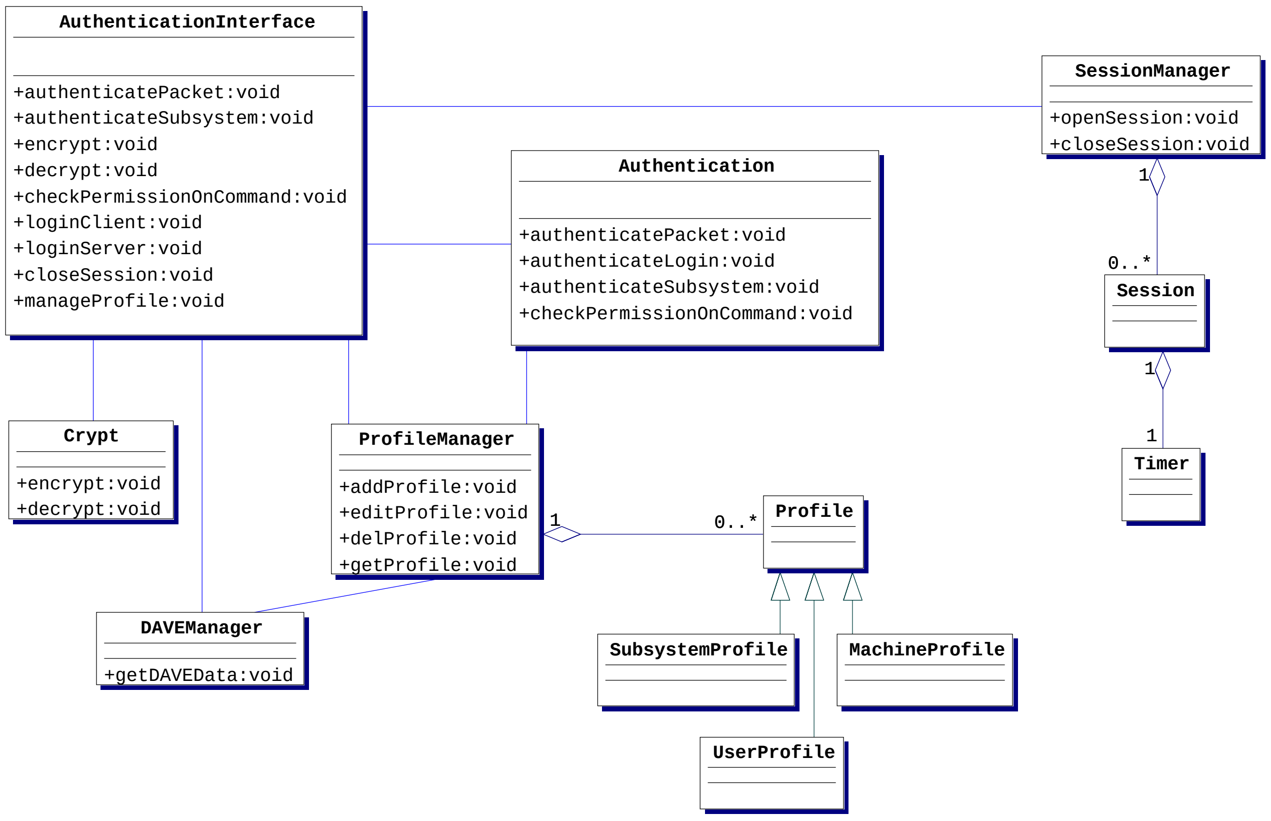

Client
Server
Timer
CloseSession
EventService

Administration Use Cases



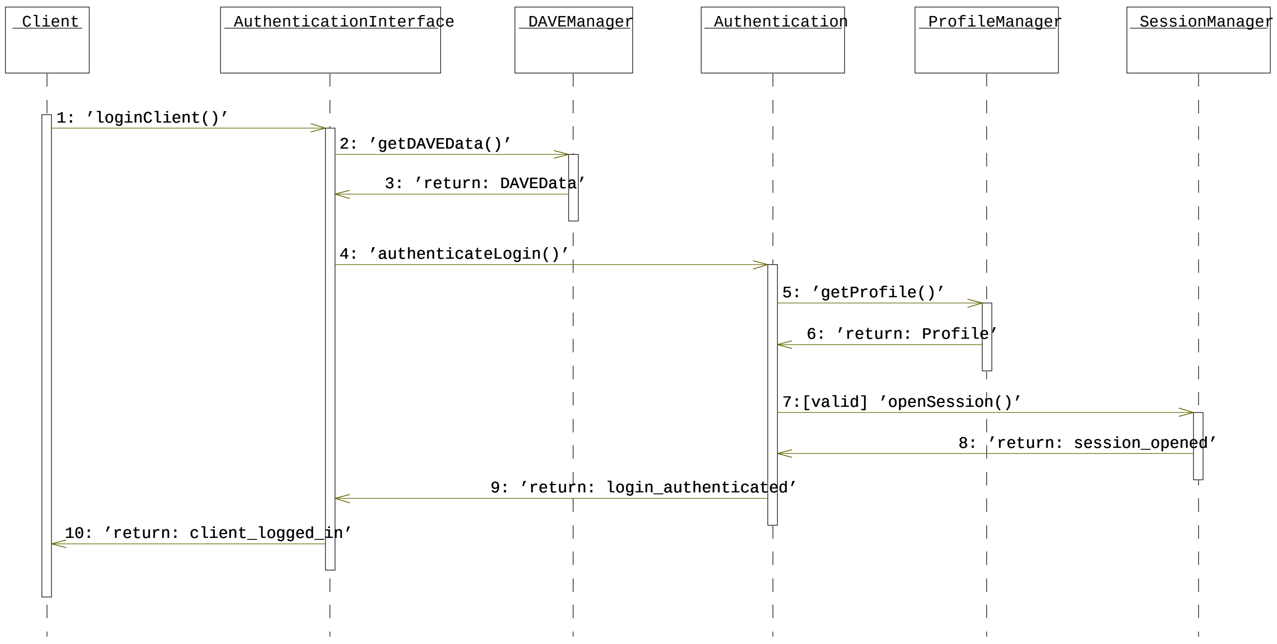
3.1.5.3 Object Models

Class Diagrams

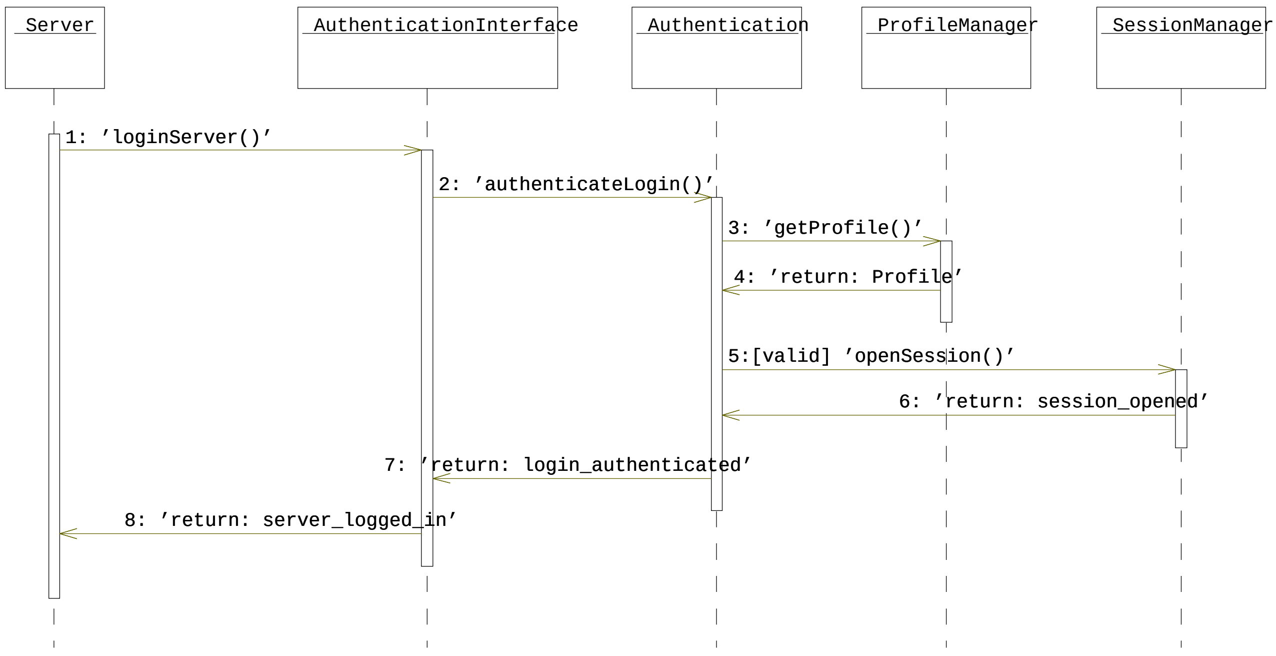


3.1.5.4 Dynamic Models

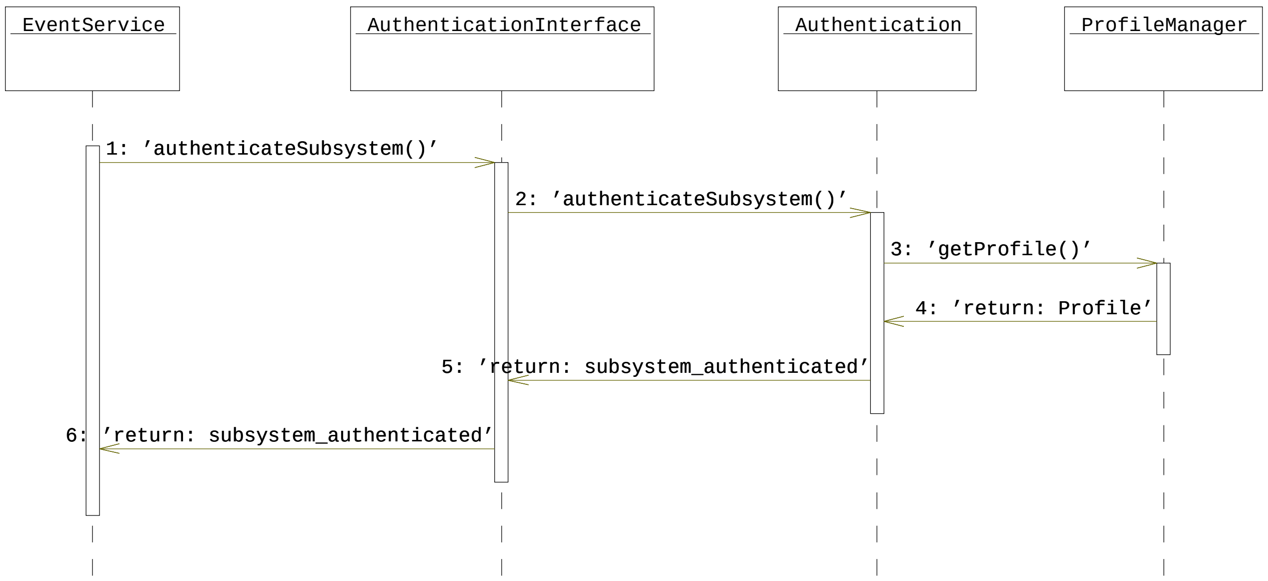
ClientLogin Sequence:



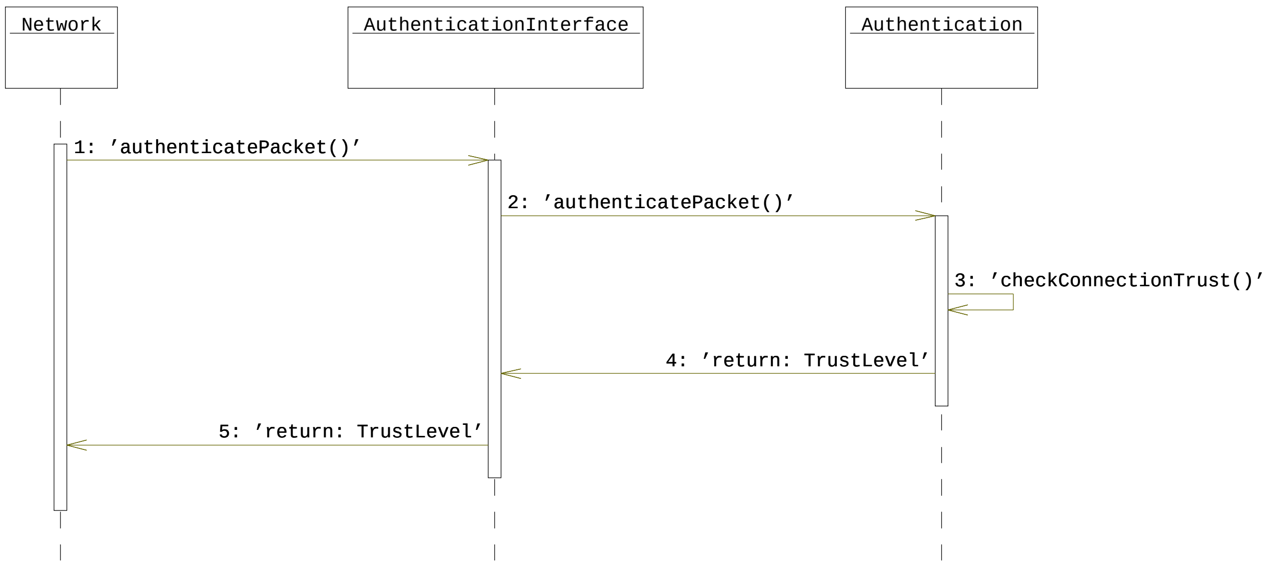
ServerLogin Sequence:



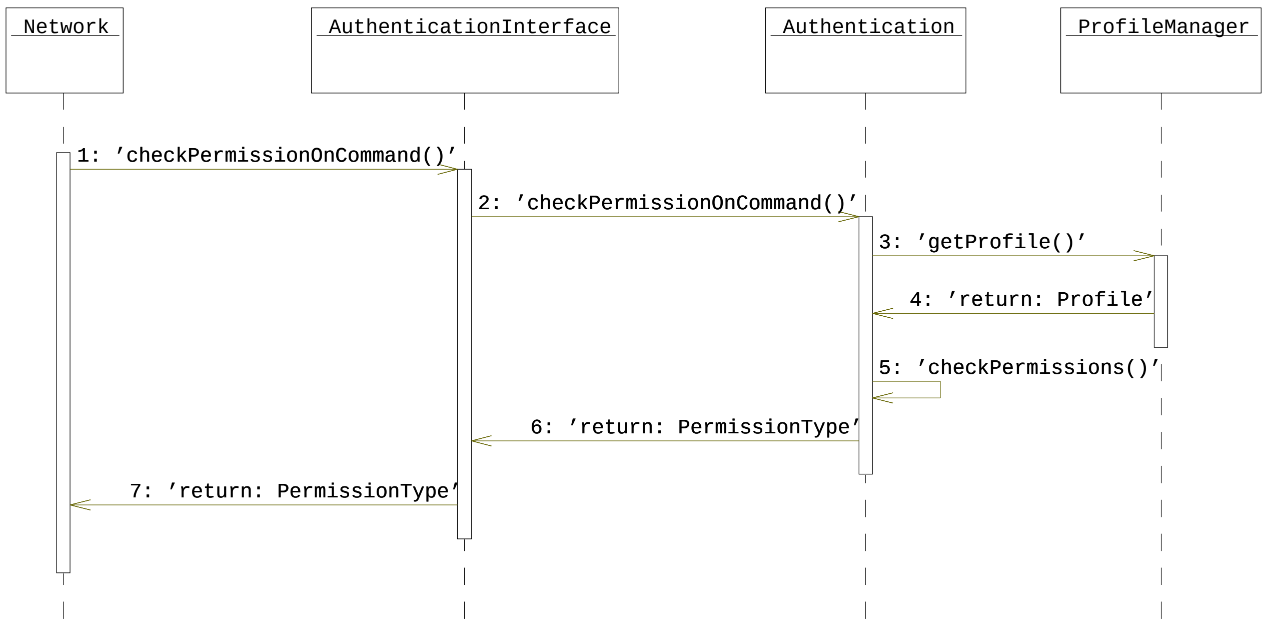
AuthenticateSubsystem Sequence:


EventService
AuthenticationInterface
Authentication
ProfileManager
1: 'authenticateSubsystem()'
2: 'authenticateSubsystem()'
3: 'getProfile()'
4: 'return: Profile'
5: 'return: subsystem_authenticated'
6: 'return: subsystem_authenticated'

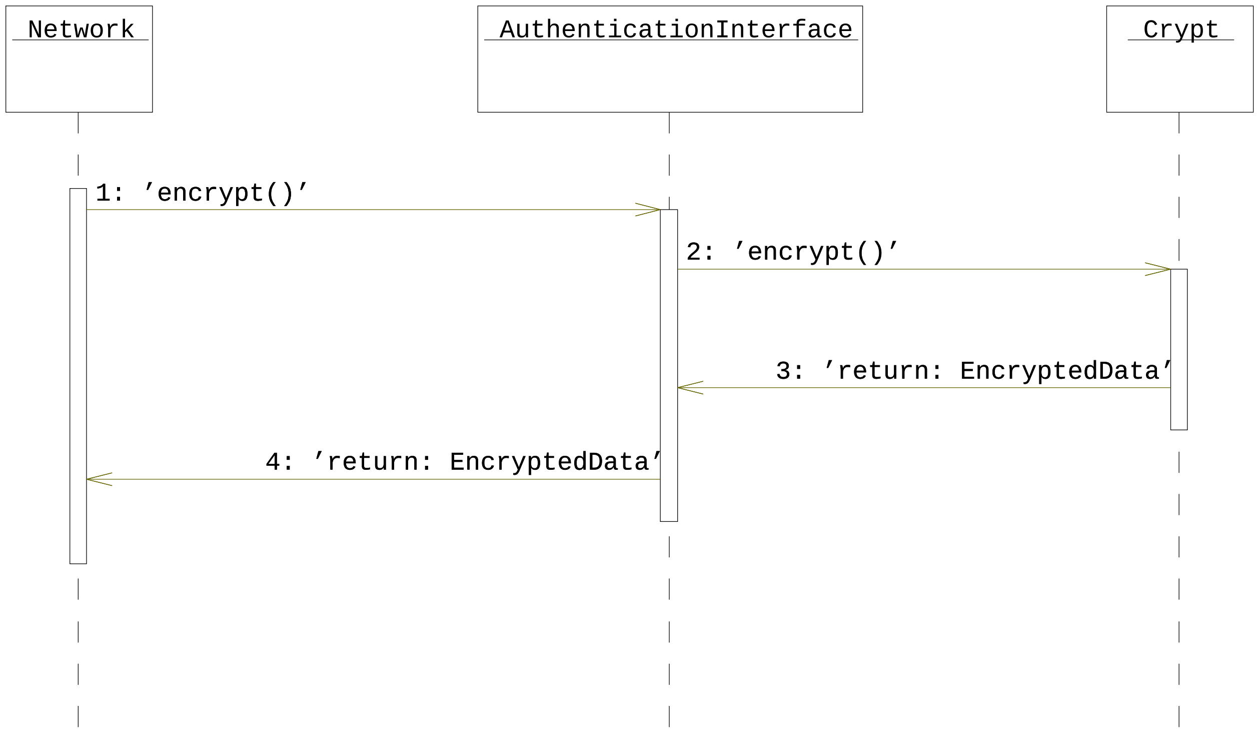
AuthenticatePacket Sequence:



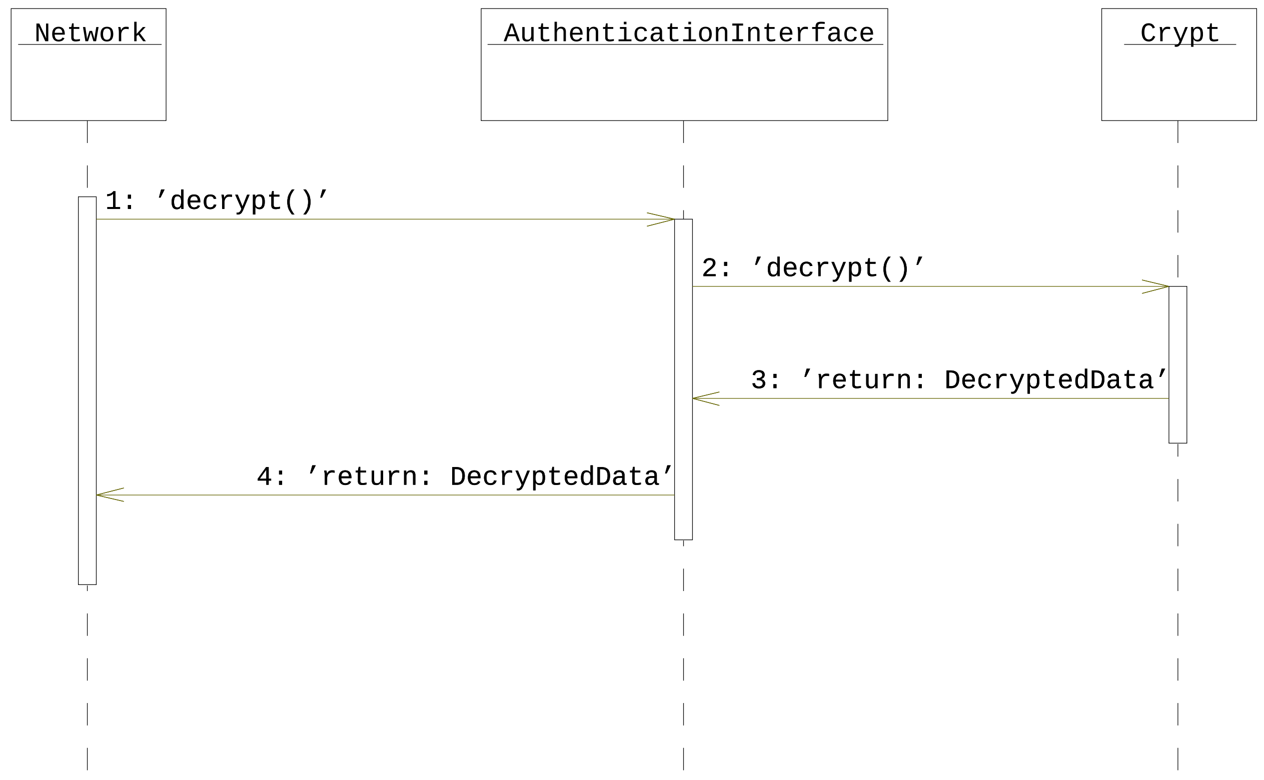
CheckPermissionOnCommand Sequence:



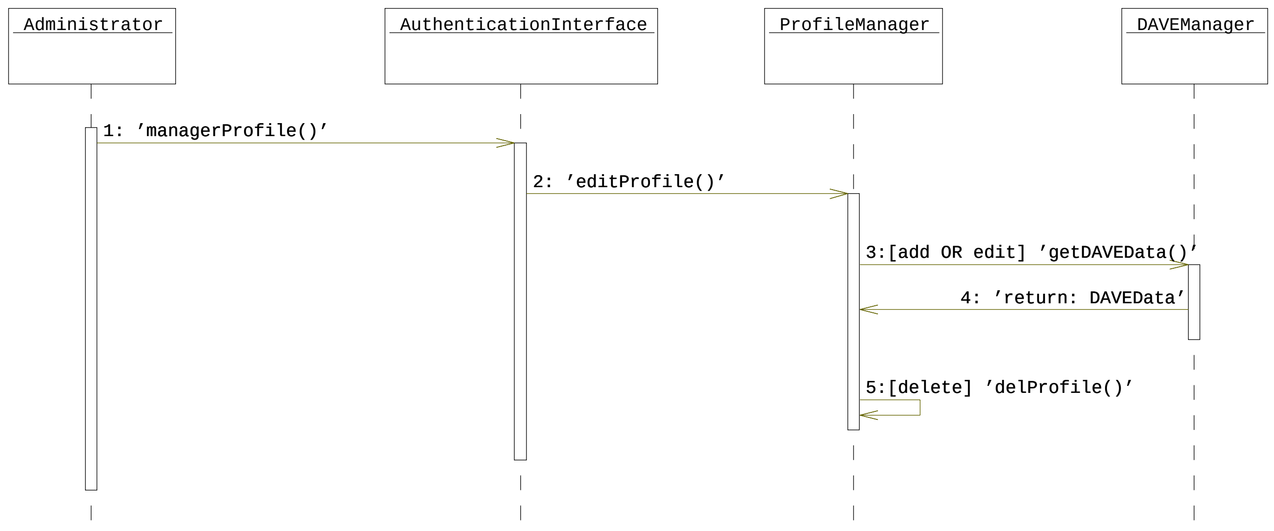
Encrypt Sequence:



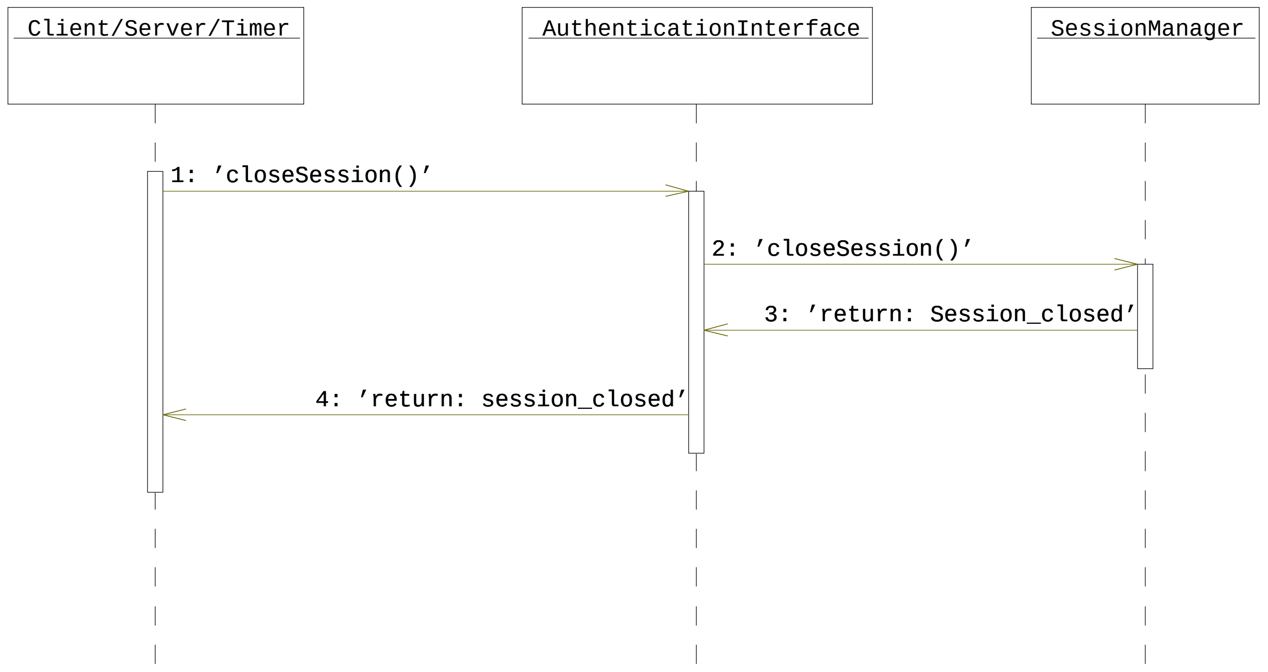
Decrypt Sequence:



EditProfile Sequence:



CloseSession Sequence:



3.1.5.5 User Interface − Navigational Paths and Screen Mockups

The Authentication subsystem is transparent to the user and therefore does not have screen mockups or navigational paths for the user to follow.

3.2 Database

3.2.1 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*.

3.2.2 Functional Requirements

3.2.2.1 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.

3.2.2.2 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.

3.2.2.3 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.

3.2.2.4 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’.

3.2.2.5 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.

3.2.3 Nonfunctional Requirements

3.2.3.1 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 *STAR NETWORK / UserInterface Subsystem's* concern.

3.2.3.2 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.

3.2.3.3 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.

3.2.3.4 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.

3.2.3.5 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.

3.2.3.6 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.

3.2.3.7 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.

3.2.3.8 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.

3.2.3.9 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.*

3.2.3.10 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'.

3.2.3.11 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.

3.2.4 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.

3.2.5 System Model

3.2.5.1 Scenarios

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.

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.*

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.*

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.

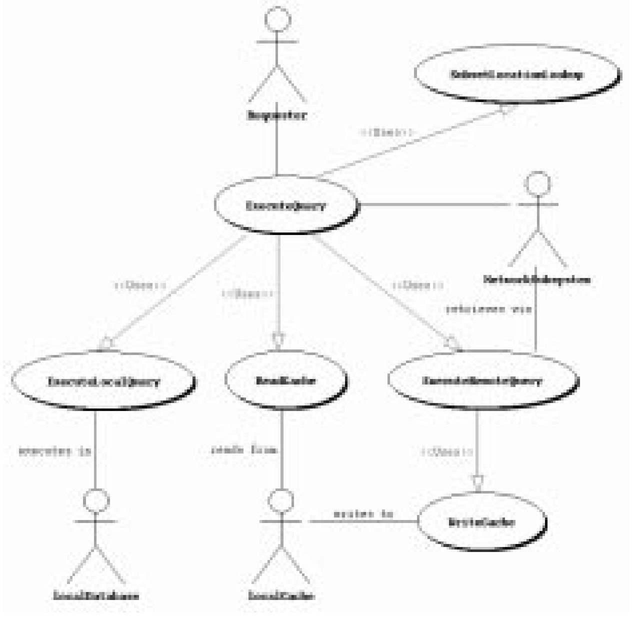
A subsystem requests a data manipulation

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.

3.2.5.2 Use Case Models

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 2. Respectively use ExecuteLocalQuery or ReadCache 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 Subsetˆ.

Use Case ExecuteLocalQuery

Entry Condition	ExecuteQuery requests locally available data
Flow of Events	1. Send SQL−Query to local Database 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 2. On success: use WriteCache to cache the ResultSet locally 3. Return ResultSet or an error

Entry Condition	ExecuteQuery requests locally unavailable uncached data
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 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

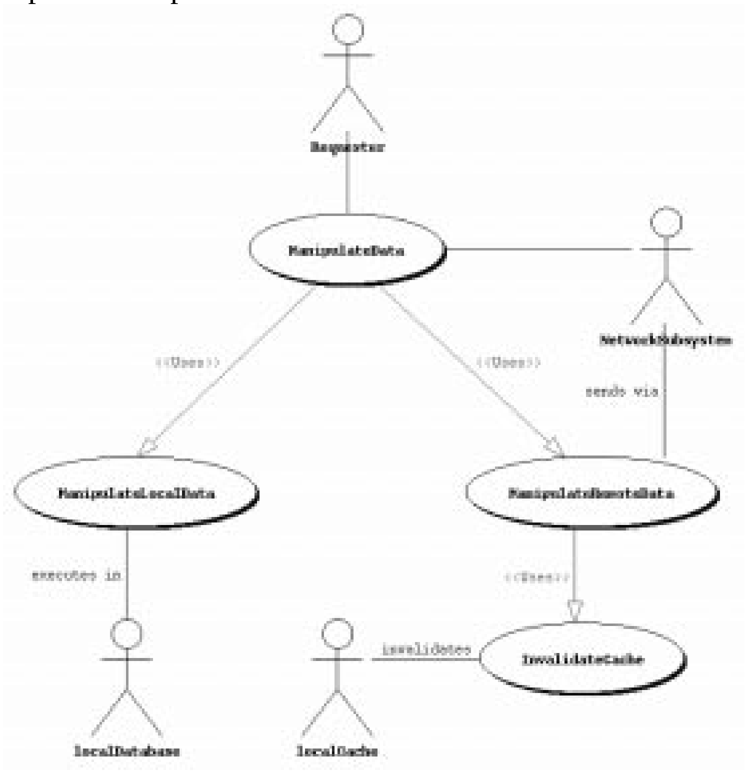
This secondary actor represents the local cache that is used by ReadCache and WriteCache.

Actor localDatabase

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

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 2. Use ManipulateLocalData in order to locally manipulate data 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 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 2. Use InvalidateCache to remove the manipulated data from the local cache 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

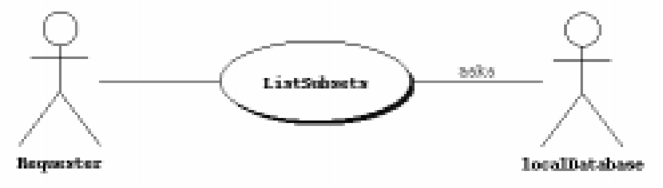
This secondary actor represents the local cache that stores recently executed queries and the results, therefore the cache entries that belong to a specific query must be removed if the data is updated.

Actor localDatabase

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

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

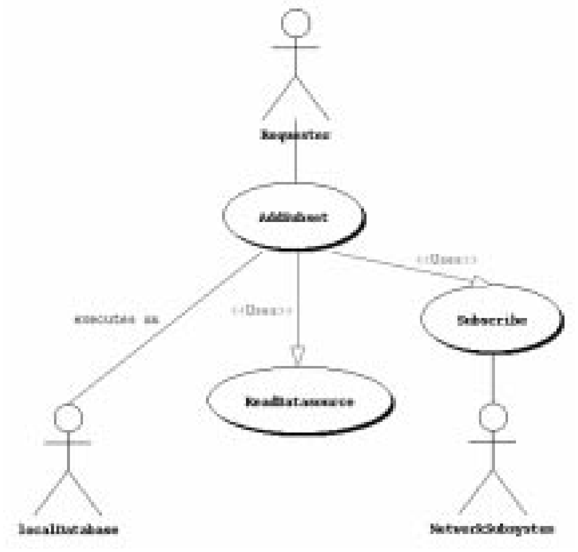
This primary actor represents a Paid subsystem that wants to know about the available subsets.

Actor localDatabase

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

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) 2. Use ReadDatasource to retrieve the subset 3. Insert SubSet in the local Database 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 Datasourceˆ.

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

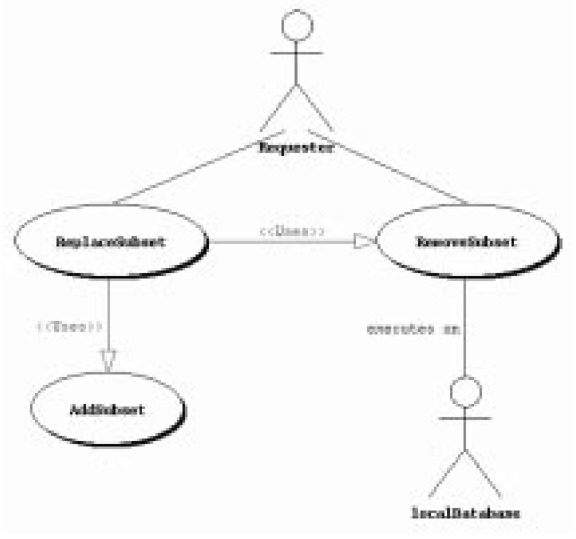
The Network Subsystem is used for the subscription of a subset.

Actor localDatabase

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

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 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 Subsetˆ.

Actor Requester

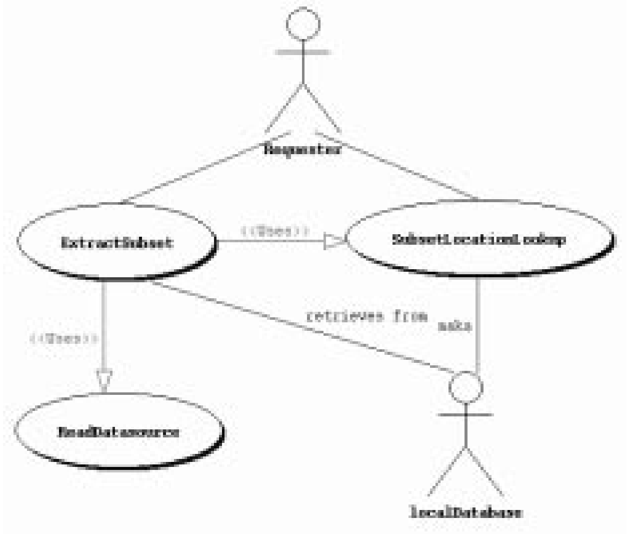
This primary actor represents the Learning, Network or User Interface subsystem.

Actor localDatabase

This secondary actor represents the local database.

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 2. If the subset is available in the local database, retrieve it 3. Otherwise use ReadDatasource to retrieve it 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 Datasourceˆ.

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 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

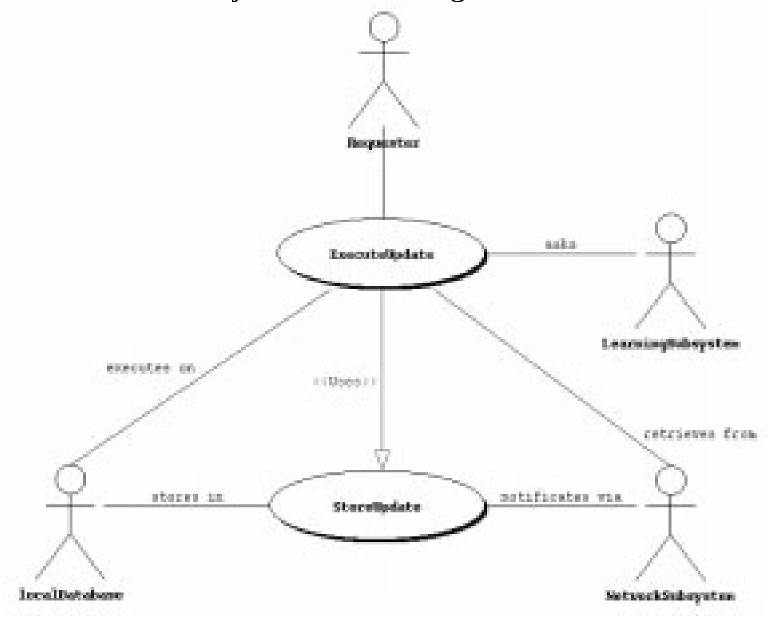
This primary actor may be the database subsystem or any other Paid subsystem.

Actor localDatabase

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

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 2. Ask Learning whether to store the update or not 3. Execute the update on the local Database 4. Use StoreUpdate in order to locally store the update
Exit Condition	Update has been executed and stored or an error condition has occurred

Entry Condition	Learning or Network Subsystem wants to execute an update, the action is already authenticated
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 2. Tell Network Subsystem to send notification to subscribers
Exit Condition	Update has been stored locally or an error condition has occurred
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

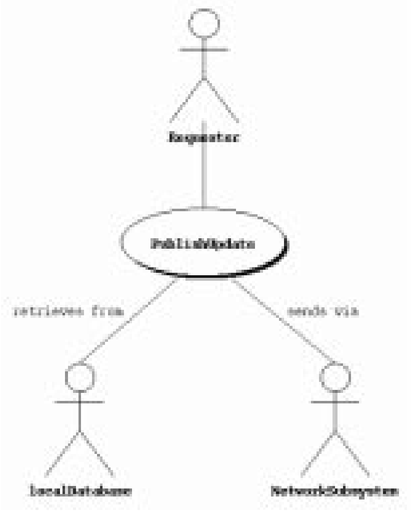
This secondary actor represents the Network subsystem that notifies the subscribers of updates or retrieves an update from the parent server.

Actor localDatabase

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

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 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

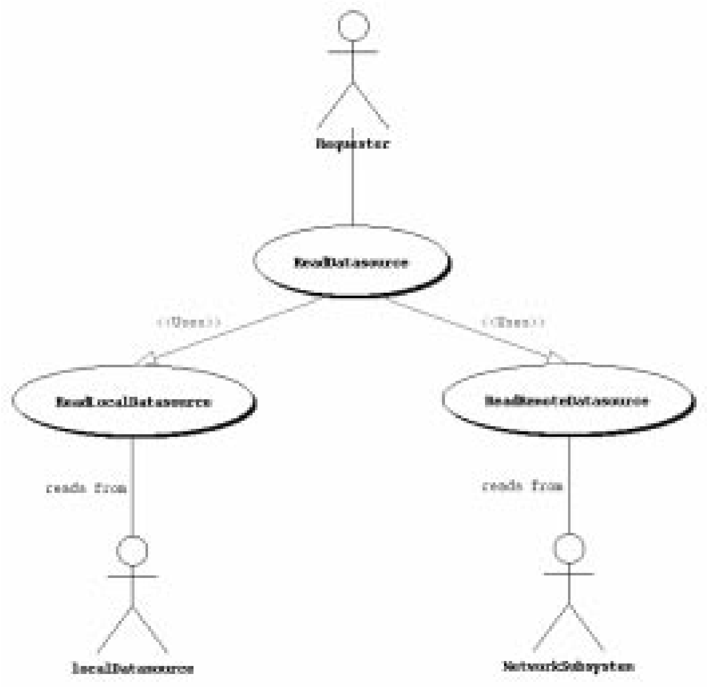
This secondary actor represents the network subsystem that is needed to send the update to the client.

Actor localDatabase

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

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 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...

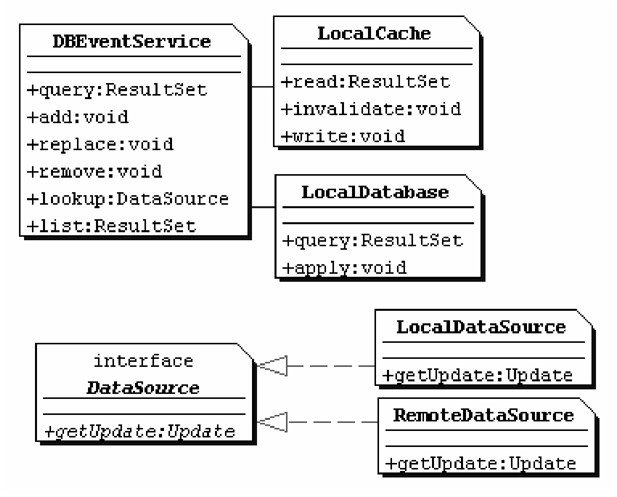
3.2.5.3 Object Models

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

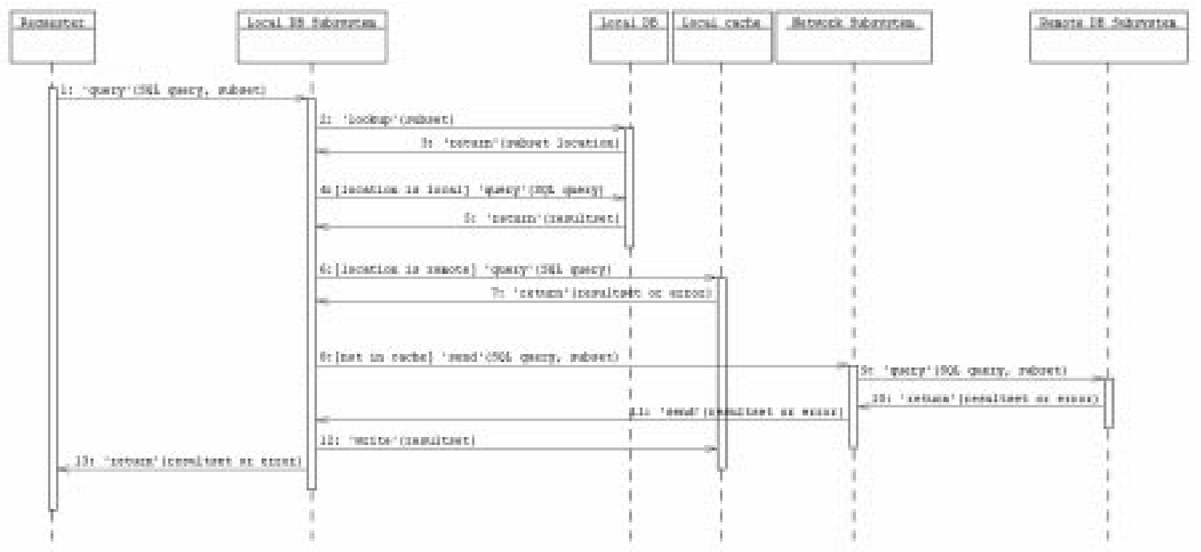
DB EventService	This will be the main class of the database API. It receives events from Network Subsystem (the event bus) and reacts on them
Remote Datasource	Acts as an interface to Network Subsystem for update requests

Class Diagrams



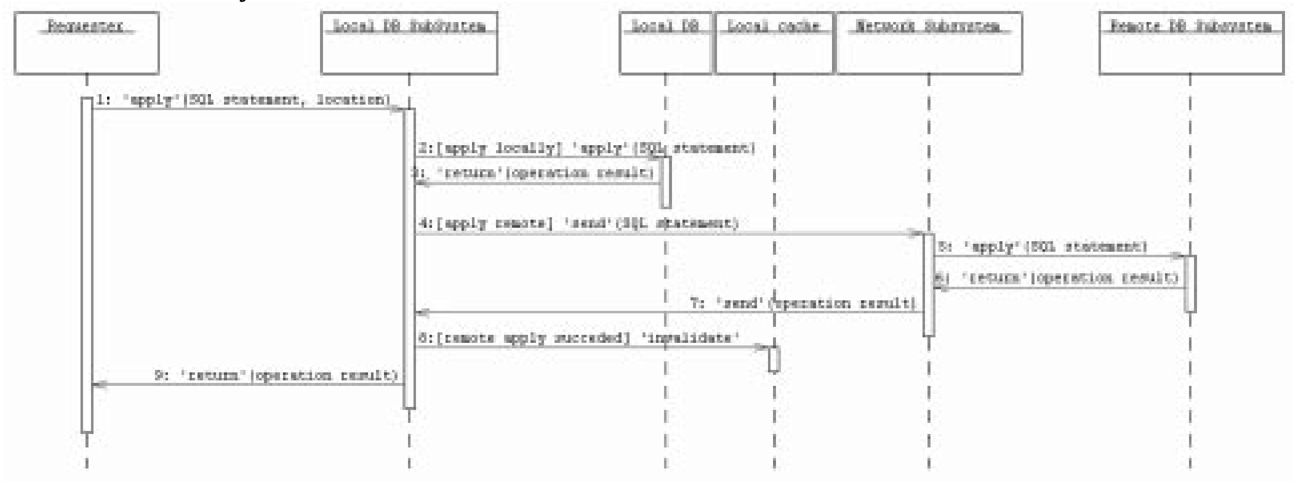
3.2.5.4 Dynamic Models

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)

1. Requester sends an SQL statement along with the subset to work on
2. Local DB Subsystem looks up the subset's location from the lookup table in Local Database
3. Local DB Subsystem receives 'local' or 'remote' as the subset's location
4. if location is 'local' the SQL query is executed on Local Database
5. Local Database gives back the ResultSet
6. if location is 'remote', Local DB Subsystem searches for this query's ResultSet in Local Cache
7. Local Cache returns ResultSet if it was already cached, or an error if not

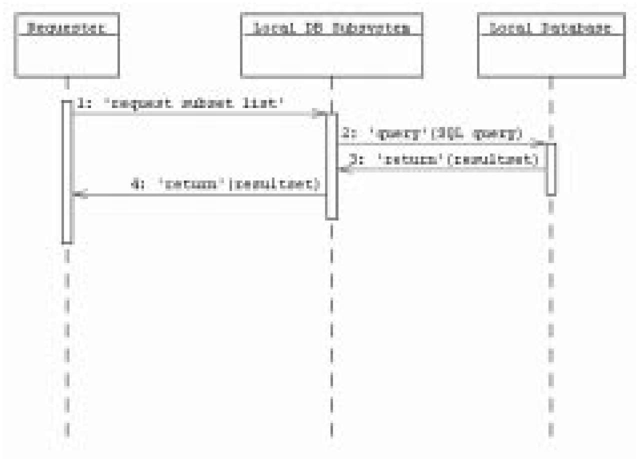
8. if the ResultSet has not been found in Local Cache, the SQL query is send to the parent server via Network Subsystem
9. Remote DB Subsystem receives the SQL statement and executes it itself
10. Remote DB Subsystem sends ResultSet or error via Network Subsystem
11. Local DB Subsystem receives ResultSet or error
12. if ResultSet was received from Remote DB Subsystem, it is written to Local Cache
13. Local DB Subsystem returns ResultSet or error


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)

Manipulate Data

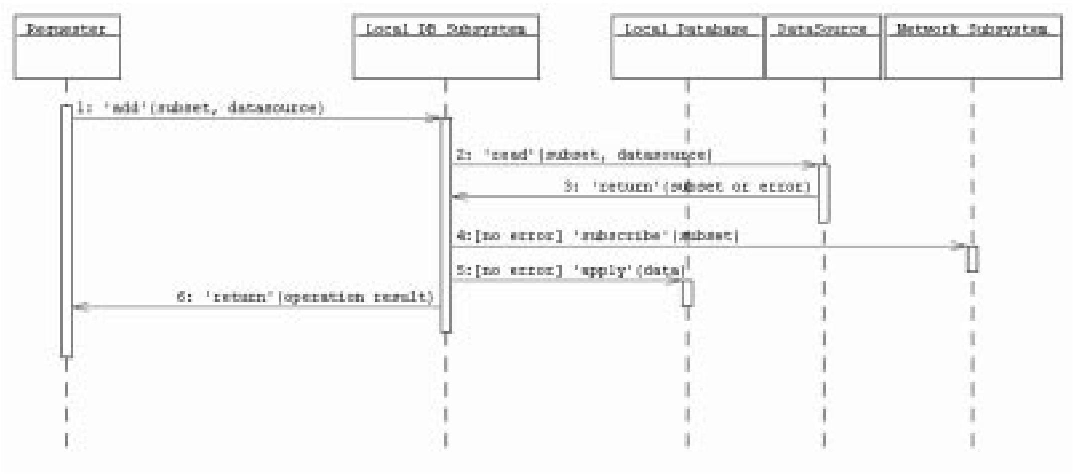
1. Requester calls Local DB Subsystem to apply an SQL statement on a given location (local, remote or both)
2. if the location is ’local’, the statement is executed on Local Database
3. Local Database gives back confirmation or error
4. if the location is ’remote’, the apply command is sent via Network 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

List subsets



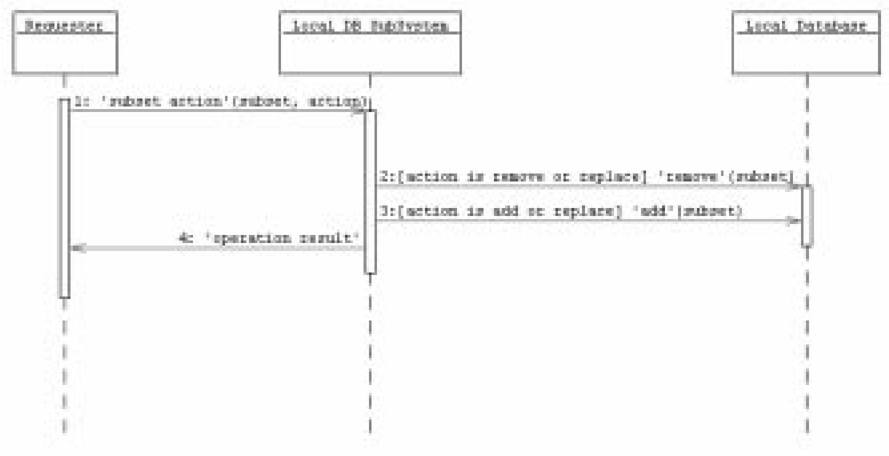
1. Requester requests subset list from Local DB Subsystem
2. Local DB Subsystem executes SQL query for all subset identifiers on Local Database
3. Local DB Subsystem receives ResultSet
4. Local DB Subsystem returns ResultSet

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)

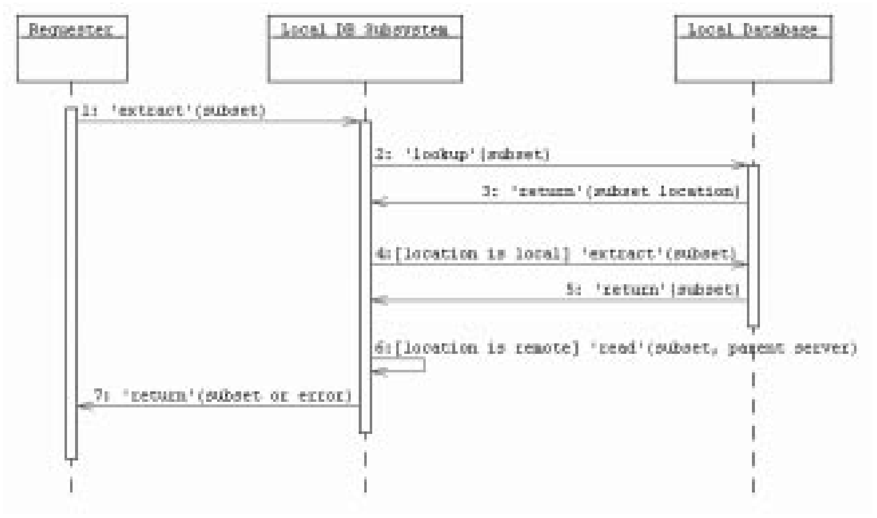
1. Requester calls add and gives the wanted subset and the datasource to read from
2. Local DB Subsystem executes ’read’ on the given datasource (see Use Case ReadDatasource)
3. Datasource returns subset or error
4. if no error occurred, Local DB Subsystem subscribes via Network Subsystem for all updates 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)

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'

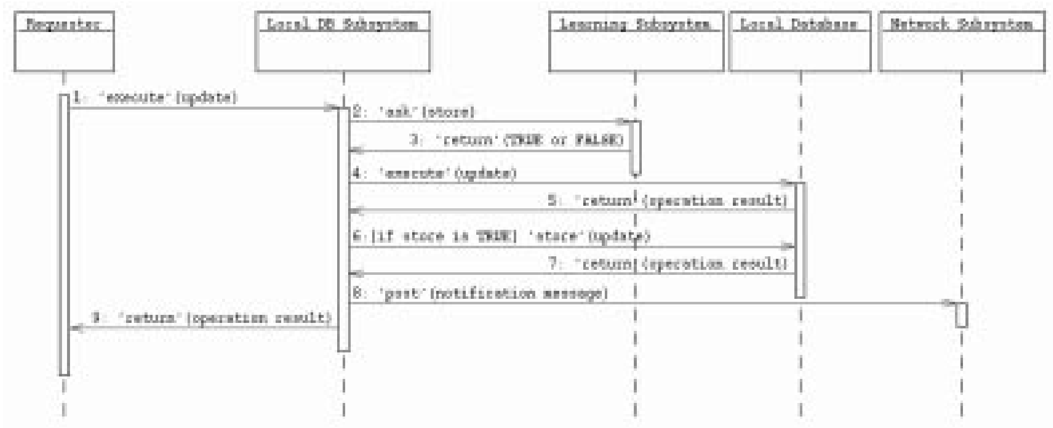
1. Requester calls ’request action’ for a specific subset (replace for two subsets)
2. if the action is ’remove’ or ’replace’, remove the (old) subset from Local Database
3. if the action is ’add’ or ’replace’, add the (new) subset (see Use Case AddSubset)
4. return confirmation or error

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)

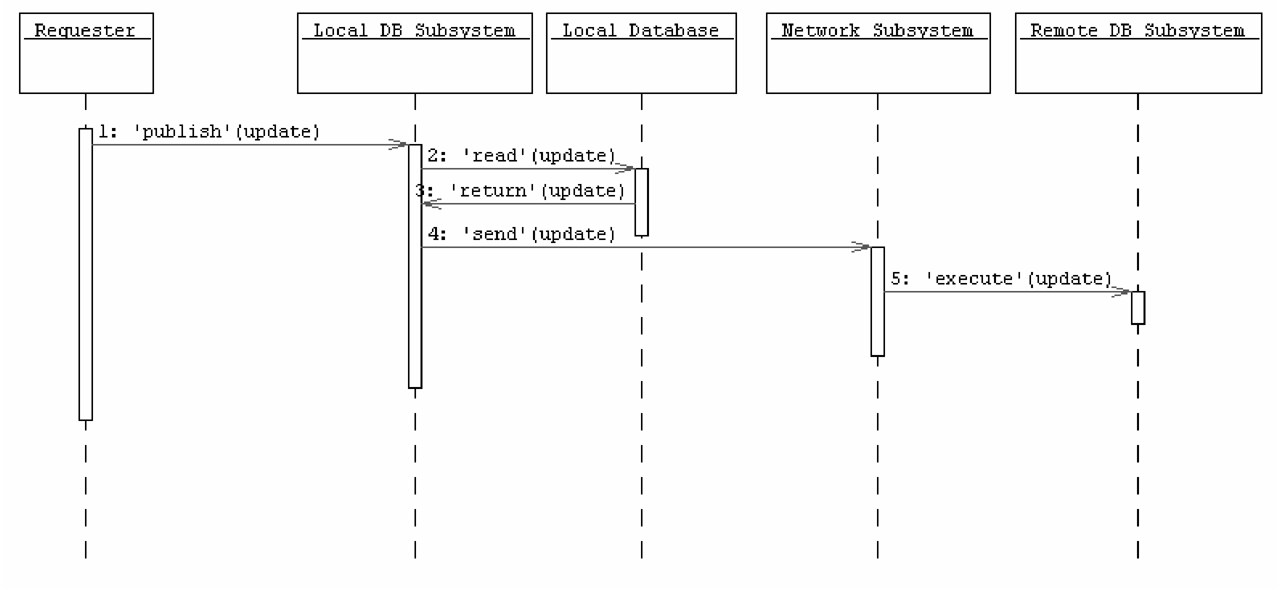
1. Requester calls ’extract’ and gives wanted subset identifier
2. Local DB Subsystem looks up the subset’s location in the lookup table of Local Database
3. Local Database returns ’remote’ or ’local’ as the subset’s location
4. if location is ’local’, the subset data is extracted from Local Database
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’)

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)

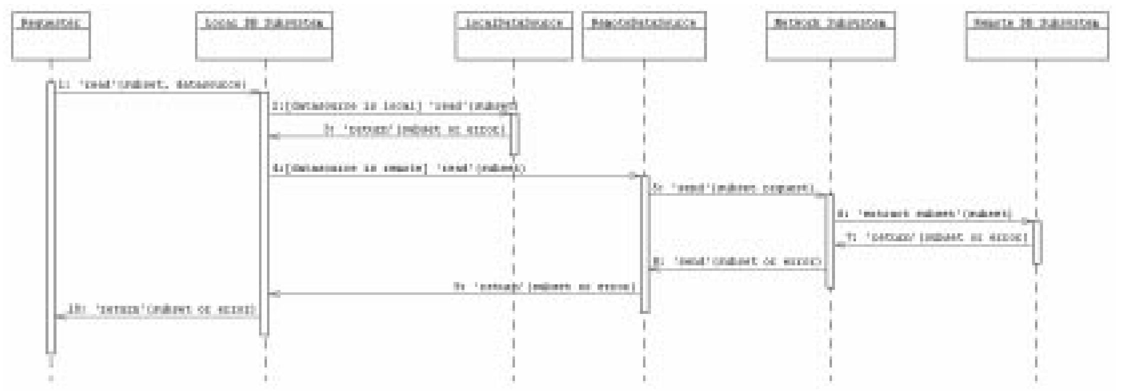
1. Requester calls ’execute’ along with the update data
2. Local DB Subsystem asks Learning Subsystem if the update data should be stored in the local update queue
3. Learning subsystem answers ’TRUE’ or ’FALSE’
4. the update is executed on Local Database
5. Local Database returns operation result
6. 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

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)

Read DataSource



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

3.2.5.5 User Interface − Navigational Paths and Screen Mockups

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

3.3 Learning and Adaption

3.3.1 Overview

The proposed learning subsystem will watch for "triggers" (data transactions) from the main Database. The system will selectively analyze this data and determine a more intelligent, streamlined and efficient way to handle future data transactions.

3.3.2 Functional Requirements

The learning subsystem will analyze behaviors and suggest actions to implement. It does this by monitoring the behavior noted by the database and responding to database triggers. Specifically, it will store frequently accessed records in the local database and reschedule updates over the network to avoid congestion and waiting lines. Second, responses to triggers from the database need to be at least semi intelligent, with a lower bound on unreasonable or erroneous prompts. Depending on the user’s specification of a cost/speed balance, the learning system should recommend only behaviors that will enhance the cost/speed performance of the system. Third, the system should be user friendly, in that it never completely take control of the system from the user. This can probably be accomplished using preference setting at the client side. The system must be able to use different algorithms at different times, depending on system parameters like cpu−usage, network traffic, etc. The logging of triggers to the local database of the dealer is recommended and has to be merged with the server logfiles. All possible information about the triggers has to be stored in the logfiles for eventually faster data mining algorithms. Finally, the decision making process should be done in a reasonably short period of time.

3.3.3 Nonfunctional Requirements

3.3.3.1 User Interface and Human Factors

There are three ways the user interacts with the learning subsystem. First, the subsystem will have a learning preferences panel on the client side. The user can use this panel to decide how much control they wish to give to the system, or how frequently they wish to be prompted for updates or changes. Second, the server can access reports of the network activity that the learning subsystem monitors through the server−side user interface it also gets access to the reports generated by the behavior files. In general, these two interfaces may be more advanced features of the entire user interface, but detailed documentation should help expedite the learning process. Finally, when the user wishes to download large files and may not desire to keep their connection open long enough to finish the download, the subsystem will let the user know of the problem. In the form of a button panel, it will inform the user of the estimated download time and give the user the opportunity to refine their request or cancel it.

3.3.3.2 Documentation

Documentation will include an explanation of both the learning preferences panel on the client side and the network activity reports on the server side. It will include a description (as examples) of the functionality that selection of the learning preferences panel options enable. Also, it will include an interpretation of the statistics generated by the network activity reports for the server interface.

3.3.3.3 Hardware Consideration

The Java based subsystem should ideally be running on any platform. Due to the fact that the subsystem will need to analyze data transactions from over 10,000 dealers (due to the Daimler−Chrysler fusion), considerable RAM will be needed to carry out the complex calculations. On the client side, the learning subsystem must pay attention to the user's connection speed and download size. It will inform the user of the estimated download time and prompt the user in the event that they wish to refine their download request. If the user's cache or memory resources are too small, the subsystem may prioritize file downloads, given that a file hierarchy is provided.

3.3.3.4 Performance Characteristics

The subsystem will perform off−line data analyzation, which will be the most time consuming process. Nonetheless, the decision making process should be relatively fast if optimal algorithms are implemented, additionally the algorithms will be exchangeable. The actions that the decision making process recommends, however, should speed up the information transaction process for the different scenarios. However, if the user's available memory or connection speed limits their downloading capabilities, the subsystem will either prompt the user to refine their request or automatically prioritize the download. Learning system should be designed for the possibility to put it on a separate machine so that it can do its work on that machine. The current intention is to pack it on the server machine.

3.3.3.5 Error Handling and Extreme Conditions

The major sections of the subsystem are persistent storage (event logs and behavior files), the data analyzation functions, and the connection to the database/event service. We assume that the persistent data is robust and stable. In the situation that we are unable to reach persistent data, we are crippled. In the event that we get too many triggers/events to process, we fall back to a less intelligent yet fast algorithm (logging, and a default response to process requests). They will be handled later by the data analyzation cycles. In the event that the data analyzation process incorrectly recommends actions, the learning preferences panel on the client side will have an option to reevaluate the behaviors recommended.

To minimize the cases where the user cannot finish their download before they disconnect, the subsystem will inform them of the estimated download time. In case of an interrupted download, there must be a mechanism, that not all of the already downloaded data has to be rejected.

3.3.3.6 System Interfacing

From the network−subsystem we need the following information:

- Actual traffic in kb/s (average value, must be provided by network itself) information about traffic must be available any time (no asynchronous communication).

- Information about past network traffic and average traffic at specific times of a day (hourly) must be available any time (no asynchronous communication) for the algorithm to decide whether or not the dataminer should switch to different processing methods (different algorithms or just simply store data).

- Receive event about high/low network traffic (asynchronous communication: receive event).

We need the following information about the local system:

- Information about the actual cpu load (no asynchronous communication).

- Receive event about high/low cpu load (asynchronous communication: receive event).

The following information is needed from the database subsystem:

- Statistics about cache, free space and used data base space is needed (no asynchronous communication).

- Receive event (log data) about download of data (meta information only) by the client (asynchronous communication: receive event).

- Receive event containing information about client accesses of the clients database to determine which group can be kept (asynchronous communication: receive event).

- receive event about updates of database (asynchronous communication: receive event)

Additionally the database subsystem is informed about recommendations generated by the learning algorithms. The learning subsystem is the only subsystem using its data, so we can safely choose any desired data format.

3.3.3.7 Quality Issues

Fundamentally, the subsystem will attempt to recommend near optimal actions. For it to be reliable, data transactions will need to be substantially more streamlined and efficient for each of the applicable scenarios. The system needs to realize when it is making erroneous recommendations and correct the process, either by user input or a system checking process. In cases where the network goes down or there are database malfunctions rendering the system unable to access new data, the system will recognize this and not act until connections are restored. Due to the fact that the data analyzation process restarts at arbitrary time intervals, if the subsystem itself were to crash it will easily resume operations once restarted. Data files will be continually saved to insure this.

3.3.3.8 System Modifications

The data analyzation process is a good candidate for future modifications. If the subsystem relies on one major data mining algorithm to analyze data, multiple, more optimal data mining algorithms for specific scenarios may be added. Using strategic patterns, this will be possible at runtime. The object oriented system will be easily extendable to accommodate such modifications.

3.3.3.9 Physical Environment

The subsystem resides on the server side, with no interaction with a physical environment.

3.3.3.10 Security Issues

The internal subsystem needs to deal with no authentication or security issues to deal with.

3.3.3.11 Resource Issues

The learning subsystem needs to rely on persistent data storage from the database subsystem. The datamining algorithm needs sufficient cpu−power.

3.3.4 Constraints

The learning problem is essentially an issue of data mining. The resources needed to carry out the calculations are the greatest concern to the system. The Java development environment is needed. Finally, domain specific knowledge may be limited.

3.3.5 System Model

3.3.5.1 Scenarios

Sam’s Busy Workshop (Scenario 3)
Participating Actor Instances

eventDispatcher: EventServices
samUpdateEvent, samBehaviorEvent: Event
samRecord: EventRecord
dealerUpdateLog: LogDB
behaviorFile: LearnedBehavior

Flow of Events

1. (Entry) The event dispatcher posts Sam?s update event on a subscribed channel.

2. Sam’s record notes this event and updates itself in the dealer update log.

3. Sam’s record also orders the behavior file to recommend an action.

4. (Exit) Based on previous events it has learned, the behavior file publishes a behavior event specialized to Sam’s needs to the event dispatcher.

Bratt’s Impatient Customers (Scenario 3)

Participating Actor Instances

eventDispatcher: EventServices
brattDelayUpdateEvent, brattUpdateReminder: Event
brattRecord: EventRecord
dealerUpdateLog: LogDB
behaviorFile: LearnedBehavior

Flow of Events

1. (Entry) The event dispatcher posts Bratt?s "delay−update" event on a subscribed channel.

2. Bratt’s record notes this event and updates itself in the dealer update log.

3. Bratt’s record also orders the behavior file to recommend an action.

4. (Exit) Based on previous events it has learned, the behavior file will later publish an update reminder to the event dispatcher.

Klaus and the M−Class (Scenario 4)

Participating Actor Instances

eventDispatcher: EventServices, Scheduler
klausInfoEvent, klausStatusQuoEvent, klausStoreLocalEvent: Event
klausRecord: EventRecord
dealerUpdateLog: LogDB
dealerInfoMiner: DataMiner
alarmClock: Scheduler
behaviorFile: LearnedBehavior

Flow of Events (Early accesses)

1. (Entry) The event dispatcher posts Klaus?s information request event on a subscribed channel.

2. Klaus’ record notes this event and updates itself in the dealer update log.

3. Klaus’ record also orders the behavior file to recommend an action.

4. (Exit) Based on previous events it has learned, the behavior file publishes an event recommending no change in Klaus’ local storage to the event dispatcher.

Flow of Events (Data mining)

1. (Entry) The scheduler wakes up the dealer information mining agent.

2. The data miner analyzes the dealer update log.

3. (Exit) Based on its analysis, the data miner updates the recommendations in the behavior file.

Flow of Events (Later access)

1. (Entry) The event dispatcher posts Klaus?s information request event on a subscribed channel.

2. Klaus' record notes this event and updates itself in the dealer update log.

3. Klaus' record also orders the behavior file to recommend an action.

4. (Exit) Based on previous events it has learned, the behavior file publishes an event, recommending local storage of the M−class information for Klaus, to the event dispatcher.

Sam's Free Connection (Scenario 5)

Participating Actor Instances

eventDispatcher: EventServices
samEvent, samBehaviorEvent: Event
samRecord: EventRecord
dealerUpdateLog: LogDB
behaviorFile: LearnedBehavior

Flow of Events

1. (Entry) The event dispatcher posts Sam?s event on a subscribed channel.

2. Sam's record notes this event and updates itself in the dealer update log.

3. Sam's record also orders the behavior file to recommend an action, providing the information that Sam has an inexpensive network connection.

4. (Exit) The behavior publishes a recommendation to the event dispatcher that does not attempt to minimize connection costs.

Klaus' Expensive Connection (Scenario 5)

Participating Actor Instances

eventDispatcher: EventServices
klausEvent, klausBehaviorEvent: Event
klausRecord: EventRecord
dealerUpdateLog: LogDB
behaviorFile: LearnedBehavior

Flow of Events

1. (Entry) The event dispatcher posts Klaus?s event on a subscribed channel.

2. Klaus' record notes this event and updates itself in the dealer update log.

3. Klaus' record also orders the behavior file to recommend an action, providing the information that Sam has an expensive network connection.

4. (Exit) The behavior publishes a recommendation to the event dispatcher that attempts to minimize connection costs.

3.3.5.1.6 Bratt's Mobile Garage (Scenario 6)

Participating Actor Instances

eventDispatcher: EventServices
frankInfoEvent, mobileInfoEvent: Event
truckMobileFixRecord: EventRecord
mobileRepairLog: LogDB
behaviorFile: LearnedBehavior

Flow of Events

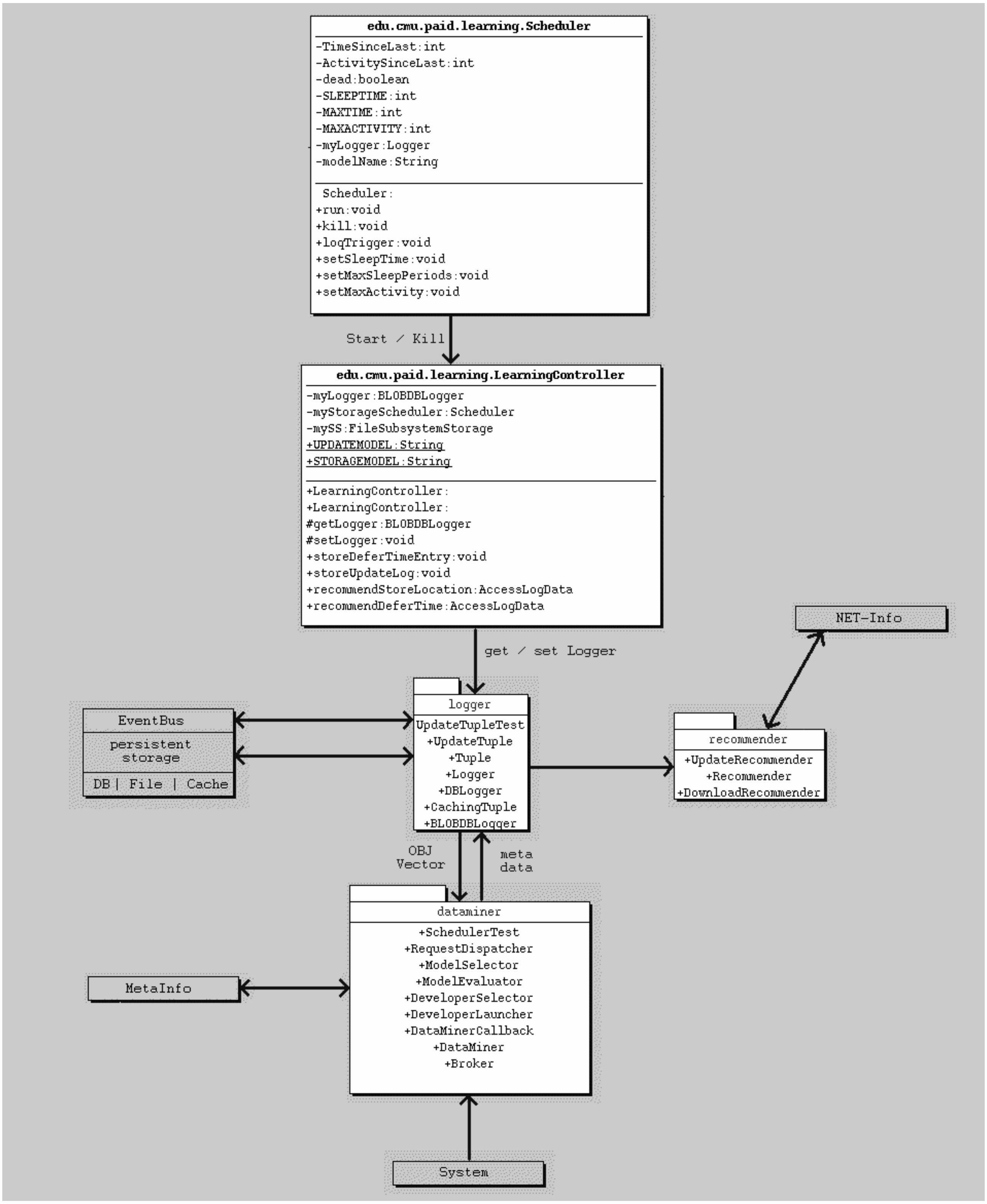
1. (Entry) The event dispatcher posts Bratt's request for Frank's truck information on a subscribed channel.

2. The mobile repair record for that type of truck notes this event and updates itself in the mobile repair log.

3. Sam's record also orders the behavior file to recommend an action.

4. (Exit) Based on previous events it has learned, the behavior file publishes a request for the list of records that Bratt will need to the event dispatcher.

3.3.5.2 Use Case Models



Participating actors are EventBus (database event) and the Scheduler. EventBus posts the event which is put into the log database ("update dealer log"). The dealer log requests information and puts it into the behavior file, which is created by the dataminer. The logger sends recommendations back to EventBus via the recommender. The scheduler, meanwhile, wakes up the DataMiner, which analyzes data and updates the Behavior File. If needed the dataminer can request meta−information. The learning controller chooses an appropriate logger (strategic pattern). The logger provides the dataminer with the necessary information and is the main interface to the database subsystem.

3.3.5.3 Object Models

Object Explanations:

EventBus: bus for all events on the server (Observer Pattern)

Logger: Contains many Event Records. Submits events through submitEvent() when DataMiner requests EventRecords and makes new EventRecord through LogEvent()

Data Miner: More than one exists for each scenario. Performs learning functions through analyzeBehavior(). Activated by scheduler to minimize likely expensiveness of its computation. Requests EventRecords from logDB through requestER(). Updates LearnedBehavior object after analyzing data through updateBehavior().

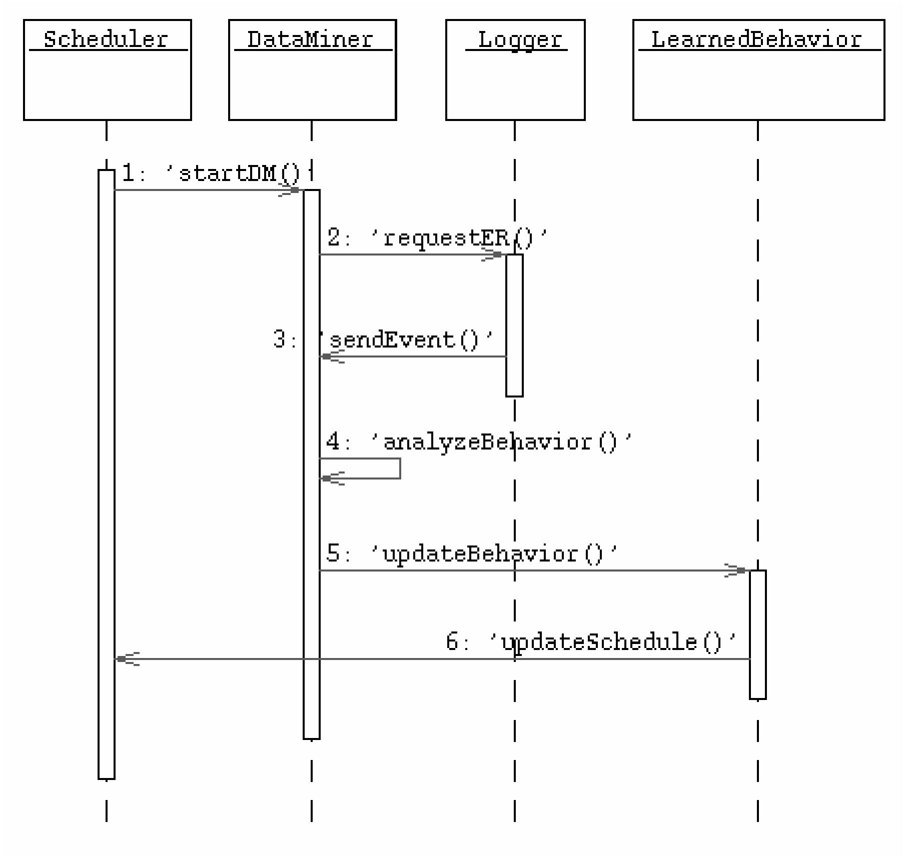
Scheduler: Activates data miner and periodic intervals through startDM()

Learned Behavior: Based on recommendations of DataMiner, sends requests to EventService through sendEvent() to perform its intelligent functions. Modifies scheduler through modifyScheduler() so scheduler can more intelligently activate DataMiner. After receiving initial request from EventServicer, dispatches it to LogDB to make a new EventRecord through postRequest()

Recommender: analyses the recommendations from the dataminer and the current net−load and then sends the recommendations via logger and event bus.

Learning Controller: chooses an appropriate Logger algorithm

3.3.5.4 Dynamic Models



The scheduler wakes up the DataMiner. The DataMiner requests an EventRecord(s) from the Logger. The Logger sends this EventRecord(s) to the DataMiner. The DataMiner analyzes the data and updates the LearnedBehavior object. LearnedBehavior then updates the Scheduler so the Scheduler can wake the DataMiner at a more intelligent time interval.

3.3.5.5 User Interface – Navigational Paths and Screen Mockups

3.4 Network and Event Services

3.4.1 Overview

The PAID Network subsystem will be designed to provide a method of communication between a dealer and the central database, as well as providing communication between the other individual subsystems of PAID. Specifically, the Network subsystem will allow a remote user to request information from the central database, and to receive updated information from the central database.

3.4.2 Functional Requirements

The Network subsystem will implement both pull and push functionality. Pull functionality will allow a dealer (on a remote machine) to request the transferral of data from the central database.

Push functionality will allow a central database to send updated information to a dealer. Also, the system will allow updates to be multi−cast to multiple dealers.

Transparently to the above mentioned functionality, the Network subsystem will provide on−the−fly compression (using a scheme such as adaptive compression) to reduce network load. Error checking will also be incorporated to ensure that data passed over the Network is uncorrupted. The Network subsystem will monitor server load and network responsivity to assure that information is routed in the most effective way possible.

3.4.3 Nonfunctional Requirements

Currently, the entire database is distributed using CD−ROM, microfiche, online, and paper methods to the various dealers.

3.4.3.1 User Interface and Human Factors

The Network subsystem will not be interacting with human users, instead the users of the Network subsystem will be interacting solely with other subsystems within PAID. The Network subsystem will provide a transparent event service with which the other subsystems can interact without regard of locality.

3.4.3.2 Documentation

Documentation will be provided that covers the specifics of the API, in terms which can be understood by the developers of the other PAID subsystems. The documentation will also cover how the Network subsystem will react under specific conditions. We will also be using JavaDoc to generate documentation from the coded implementation. All documentation, TogetherJ projects, and code, and libraries shall be managed via CVS.

3.4.3.3 Hardware Consideration

The Network subsystem does not deal with specific machine−type constraints. Since the system will be implemented in Java, the code will be usable on a variety of machines. The type of connection used (Modem, Ethernet, ATM, ISDN, GSM, APS, etc) will have an affect on the network subsystem’s operation, in that the Network subsystem will attempt to determine the speed of the connection, and adjust the network parameters (such as TCP/IP window size and compression methods) in order to make full use of the connection speed and to minimize congestion.

3.4.3.4 Performance Characteristics

The Network subsystem is expected to transmit at the highest speed allowable based on the connection between a remote machine and the central database. The estimated ratio of server to clients will be 1:150. In the event of a network overload, the network will use shared throughput and attempt to circumvent bottlenecks by transmitting packets along an alternate routes.

Response time will be dependent on how quickly congestion is discovered. We estimate 5−7 transactions during peak times and 2−3 transactions during off−peak times. By using shared throughput we will balance the server high bandwidth with vs. the dealer low bandwidth. We will be using adaptive compression to reduce network load.

3.4.3.5 Error Handling and Extreme Conditions

There are three exceptional situations that may arise. The first situation is network problems. In the case that the network becomes unreliable, any currently ongoing transmissions will be terminated, and both ends (the remote machine and the central database) will be informed of the interruption as well as the state of the transmission upon termination. The second possible situation is that the dealer (on a remote machine) may choose to terminate a transmission, in which case the central database will be informed of the situation and both ends will be informed of the state of the transmission at the time of termination. The third possibility is that the local program is terminated unexpectedly in the middle of a transfer (e.g. in case of power failure). In this case, the client would clear received information from its cache upon reboot and continue as if transfer never occurred.

3.4.3.6 System Interfacing

The network subsystem functions as the lowest layer or core of the PAID project. All messages sent over the network will be passed through this subsystem. All interfacing to this subsystem will be done through the Network API (NAPI). However, there are two main subsystems that we expect to interface with NAPI in order to provide the basic functionality of PAID. We anticipate the event/learning subsystem to pass data for automatic server to client updates through the network, in the form of either a point to point or multi−cast message. We also expect the database subsystem to request and receive desired information from the server through the network subsystem. All interactions between these subsystems will take place on the CORBA bus through a Java interface.

3.4.3.7 Quality Issues

This subsystem must provide fast, portable, and reliable information transportation over the network for the PAID system. In order to maximize speed efficiency, the network subsystem will provide compression on the fly, be selective (multi−cast or point to point) to the amount of information transported (to prevent unnecessary lags), and optimize routing of the information (in terms of time). To ensure portability, the subsystem is fully implemented in Java. This means any hardware/operating system which has a Java Virtual Machine will be able to route information through PAID over the network.

3.4.3.8 System Modifications

As more dealers are added to the system, scalability issues might have to be considered if the network's response is being depreciated by an overload of responses to the server. We do not anticipate any modifications outside of this as we expect the network subsystem to be very stable. We achieve this by using Java as our language as it is cross−platform and in wide use.

3.4.3.9 Physical Environment

This is not an issue because we are using Java technology, which is independent from the actual hardware PAID is running on.

3.4.3.10 Security Issues

All user access to data will be controlled by authentication and database subsystems. Transport over the Daimler−Benz intranet and extranet needs no special security measures. Transport over the general internet needs to be encrypted to insure security.

Physical security is important for both the database servers and for local copies of the database on dealer machines.

3.4.3.11 Resource Issues

System installation will be performed along with the rest of the application. Little to no maintenance will be required for the network subsystem after the initial setup stage. Neither will any persistent data be externally accessible.

3.4.4 Constraints

System will be developed with Java (ver. 1.1.x). Object and CASE models will be created with Together/J. Source Control will be handled using CVS.

3.4.5 System Model

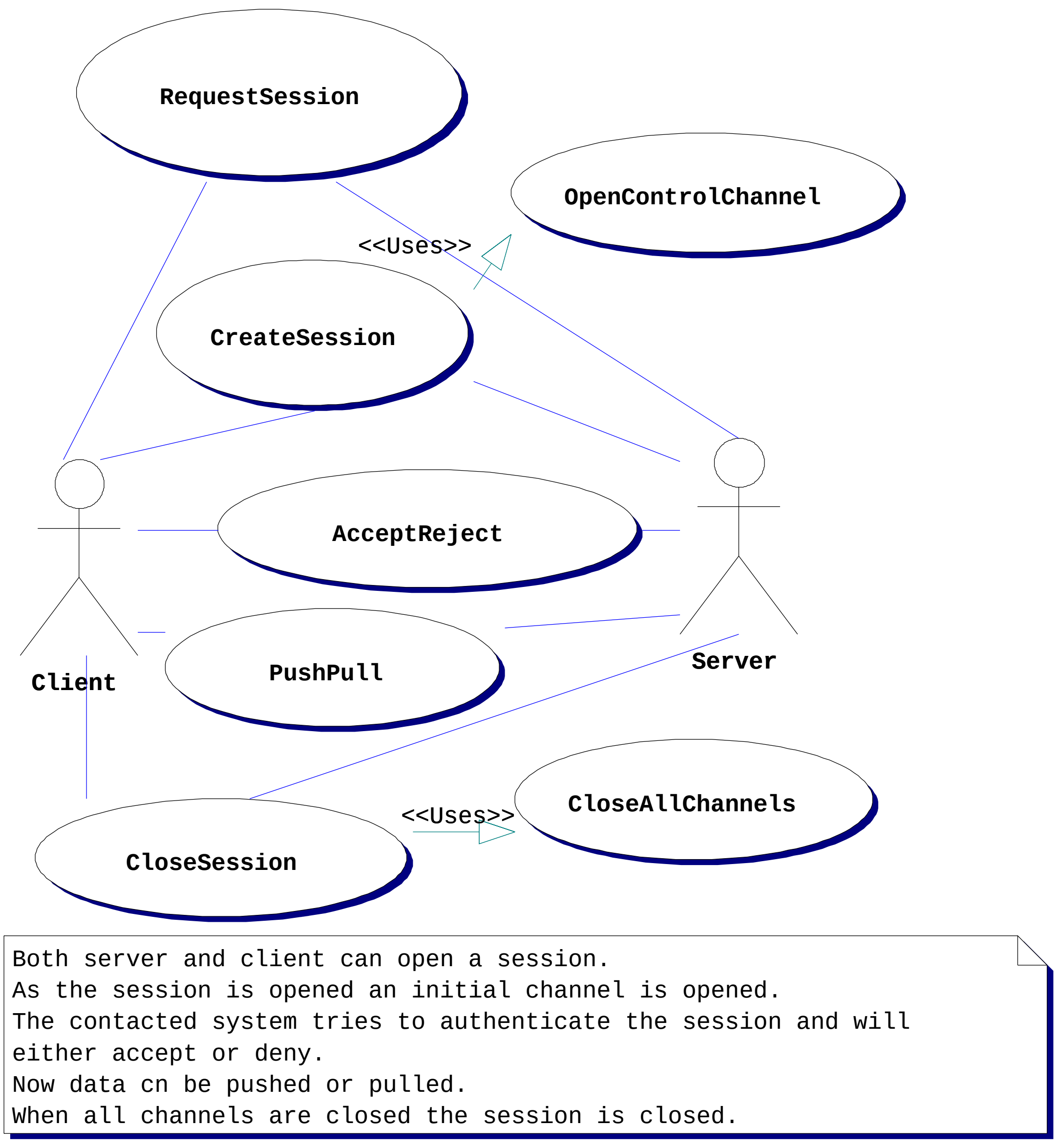
3.4.5.1 Scenarios

[Push] Mr. Benz has come out. with a new car model. He adds information about the new model to the central database in Germany. This new data needs to be transmitted to all dealers interested in this class of car. The central database tells the network to announce the availability of the new information to the relevant dealers. Bob (a dealer in San Francisco) receives the notification and requests, via the network subsystem, that the data be sent immediately. The network subsystem will be making use of the CORBA bus to remotely call procedures on the database server. The central database, upon receipt of the acceptance, now sends the updated information to Bob’s database via the network. Jim (another dealer in Miami), is busy with clients and requests that he be notified later. The network sends the request to the central database, which will log this request and later send Jim another notification.

[Pull] Humphrey, a Daimler−Benz dealer in Edinburgh, has decided to expand his model selection with a new S−class model. He logs into the PAID system, and fills out the a form to request information on the model. He clicks the submit button, and the his local database discovers that it doesn’t contain the desired information, so it passes the request to the network subsystem. The network subsystem uses CORBA bus to request information from the central database. The central database will then use the network subsystem to transmit the request information back to the database to Humphrey’s machine.

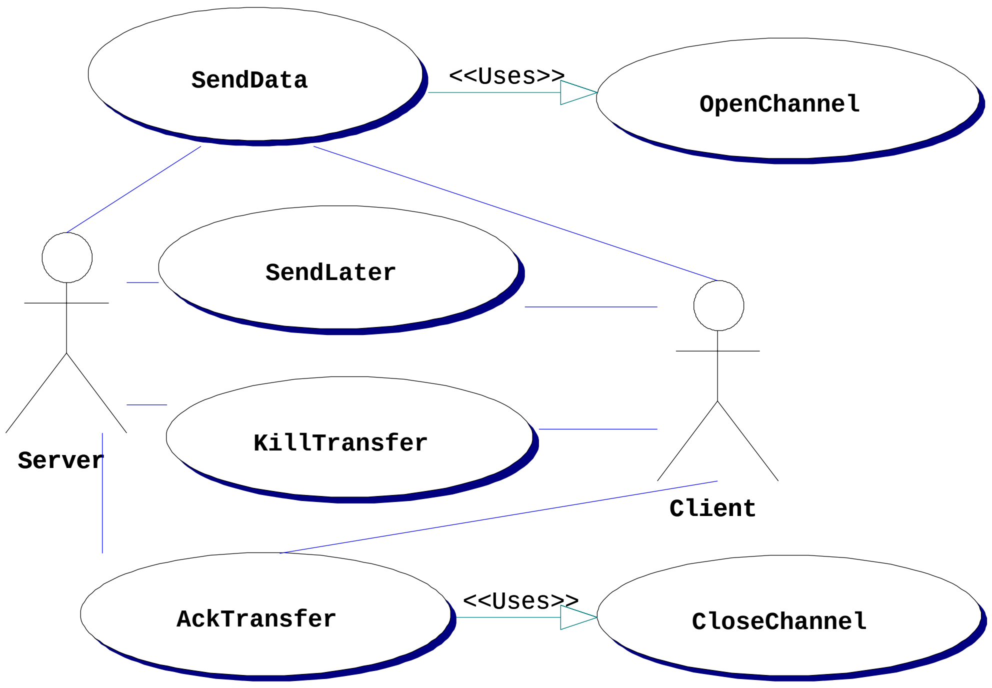
3.4.5.2 Use Case Models

Session Use Case :



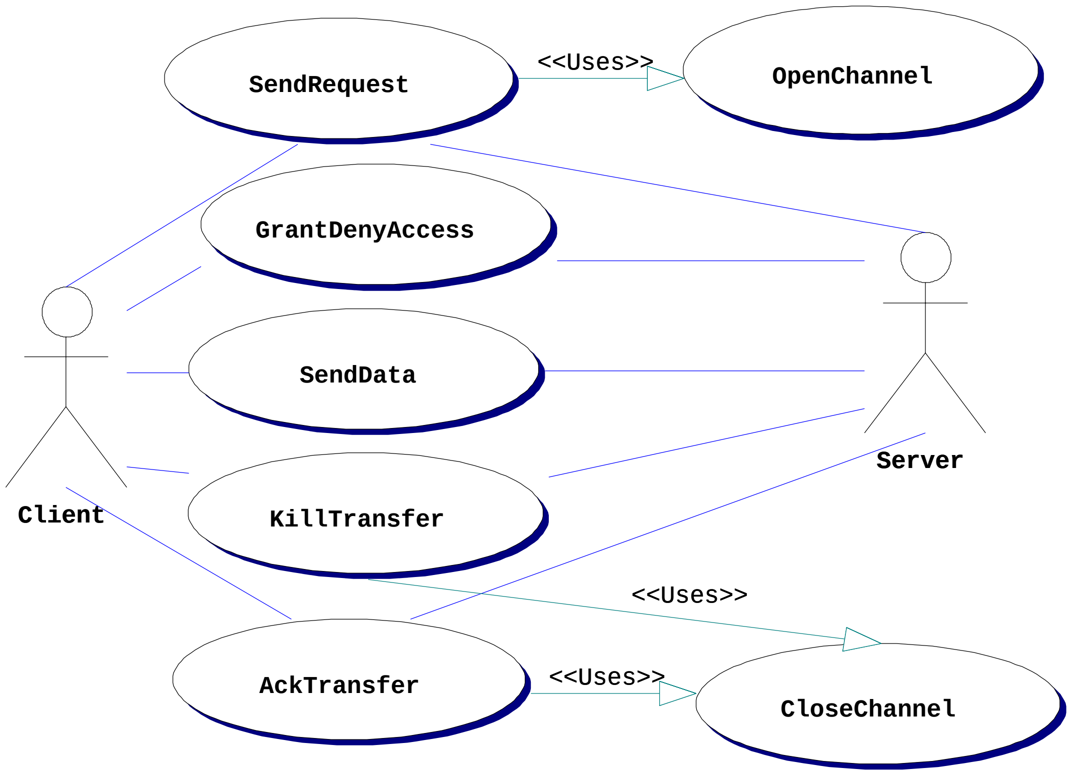
```
Both server and client can open a session.
As the session is opened an initial channel is opened.
The contacted system tries to authenticate the session and will
either accept or deny.
Now data cn be pushed or pulled.
When all channels are closed the session is closed.
```

Push Use Case :



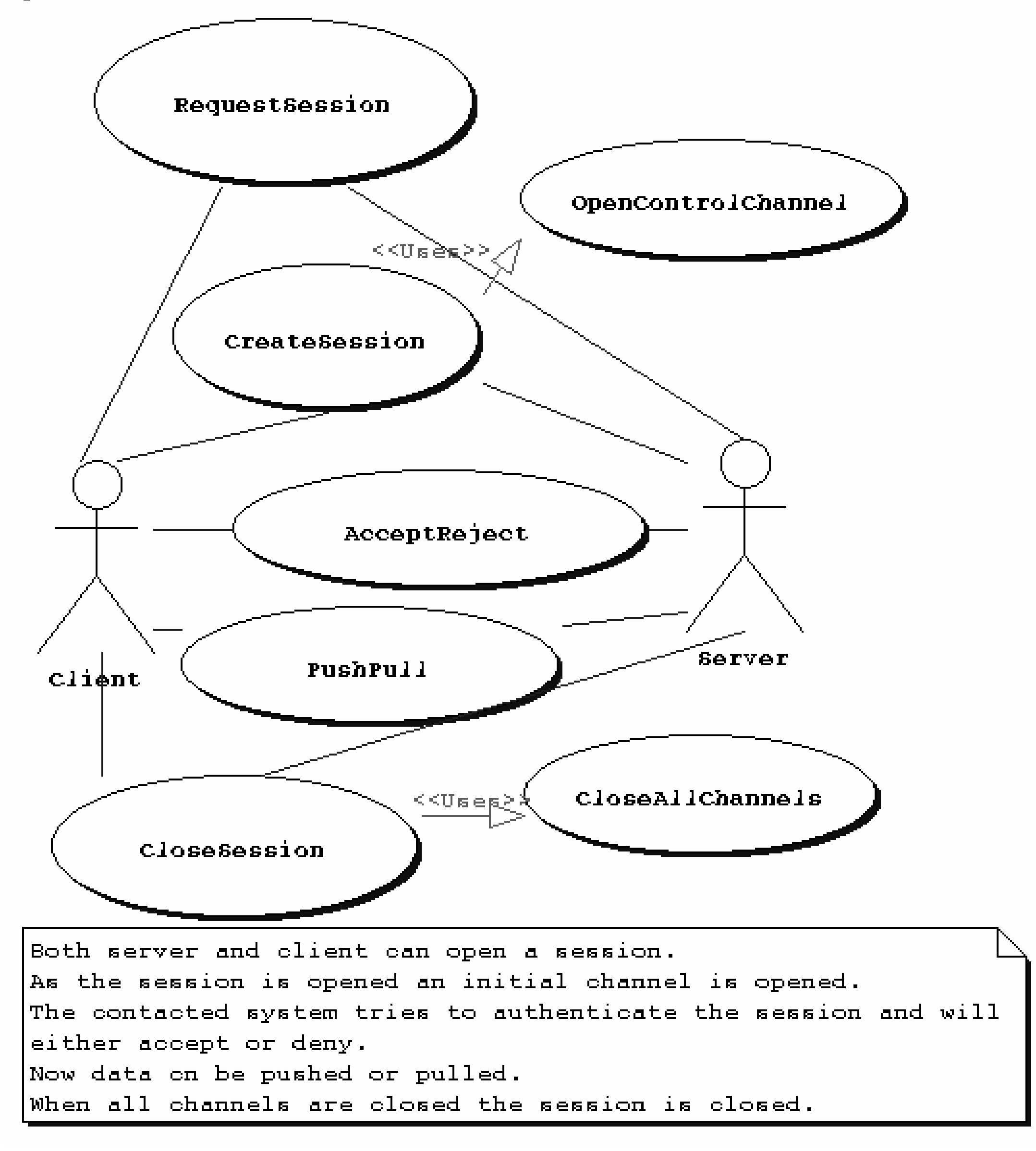
```
The pull use case is initiated by the client part of a PAID
system.
The client sends a request which opens a channel to the server.
The server may either grant or deny this request.
If access is granted the server sends the requested data.
The transfer may be killed by both sides.
The clinet acknowledges the transfer upon receipt of the
requested data.
Both KillTransfer and AckTransfer will close the channel.
```

Pull Use Case :



The pull use case is initated by the client part of a PAID system.
The client sends a request which opens a channel to the server.
The server may either grant or deny this request.
If access is granted the server sends the requested data.
The transfer may be killed by both sides.
The client acknowledges the transfer upon receipt of the requested data.
Both KillTransfer and AckTransfer will close the channel.

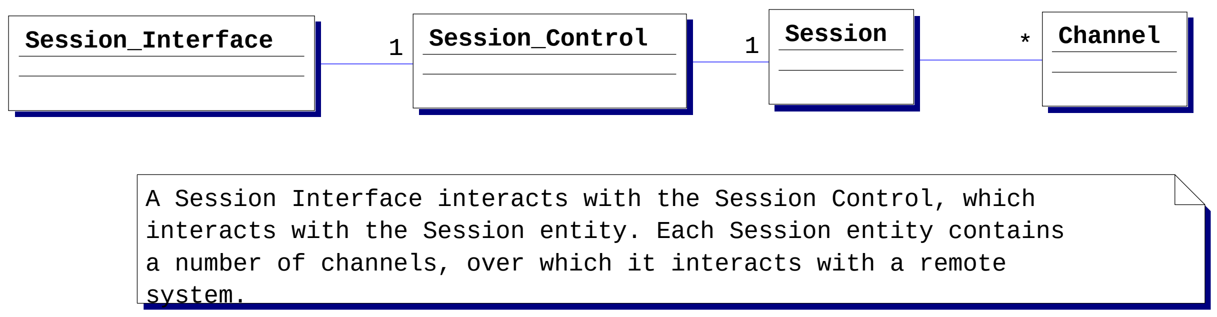
Open / Close Use Case :



Both server and client can open a session.
As the session is opened an initial channel is opened.
The contacted system tries to authenticate the session and will either accept or deny.
Now data can be pushed or pulled.
When all channels are closed the session is closed.

3.4.5.3 Object Model

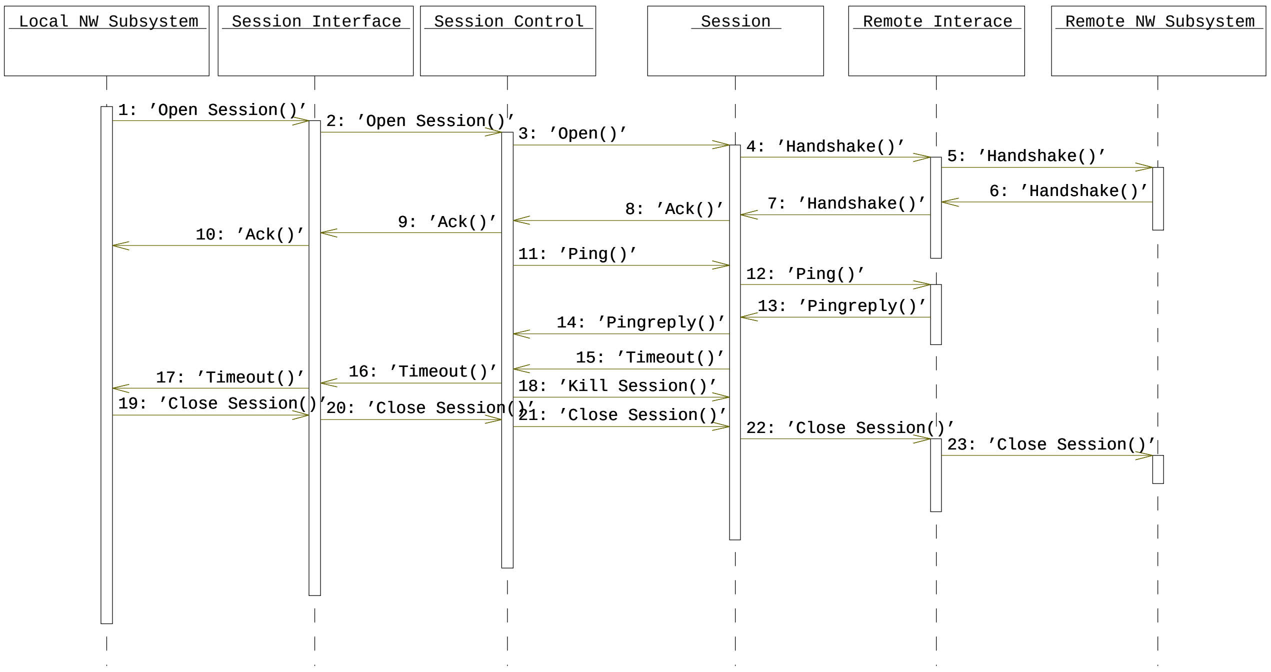
Object Model:



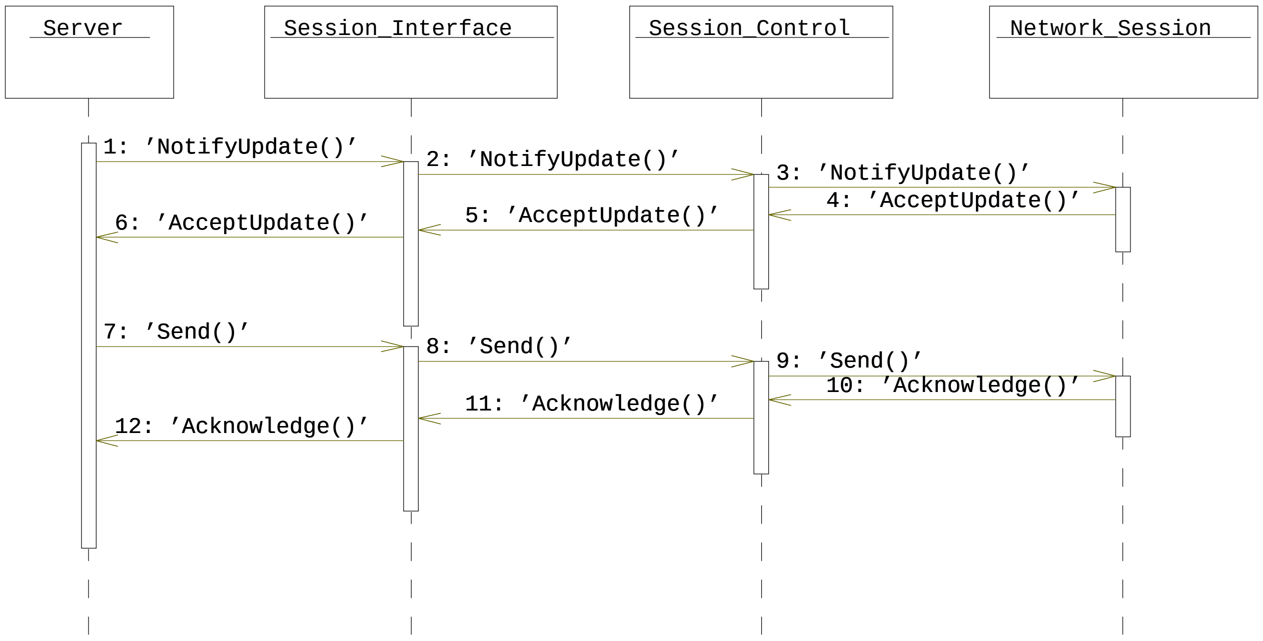
A Session Interface interacts with the Session Control, which interacts with the Session entity. Each Session entity contains a number of channels, over which it interacts with a remote system.

3.4.6 Dynamic Models

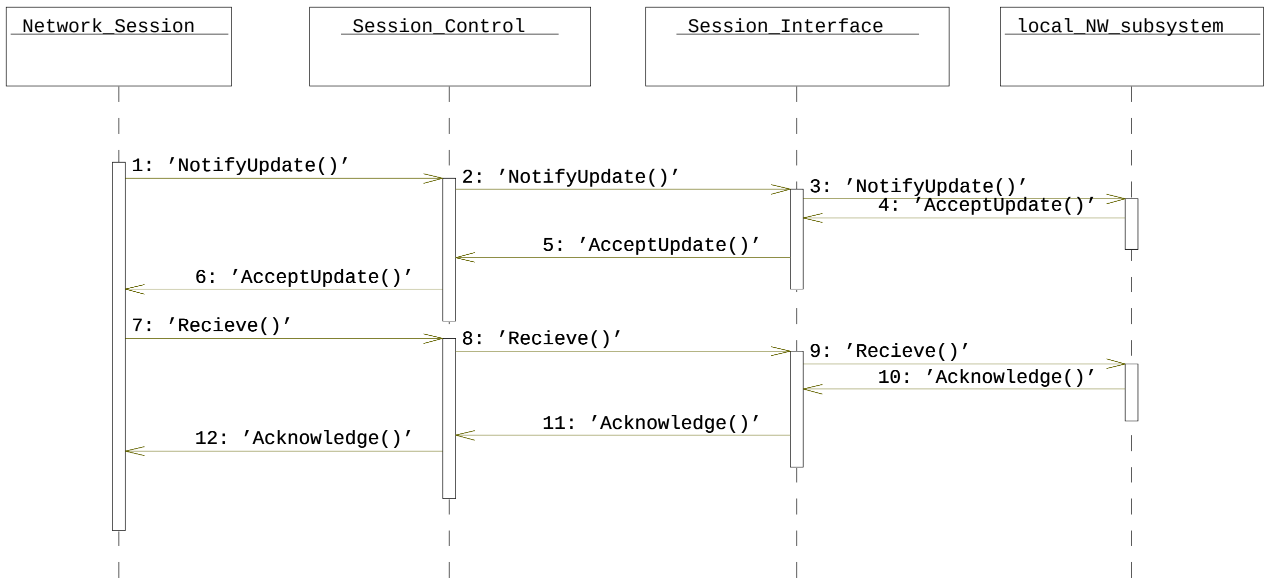
Open / Close Session :



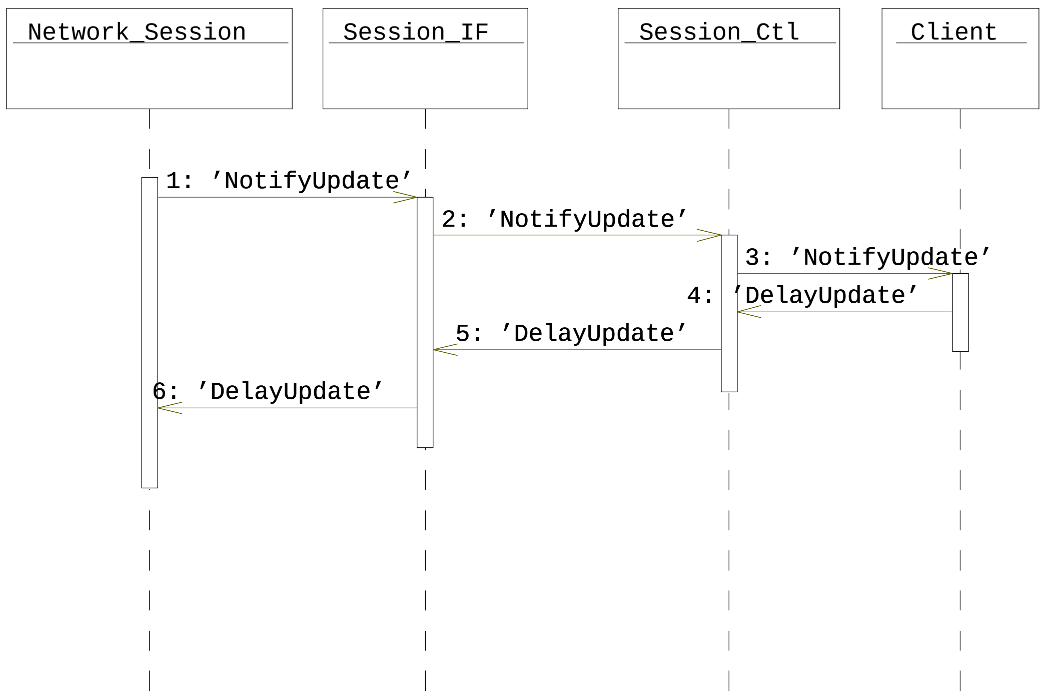
Server Side Push :



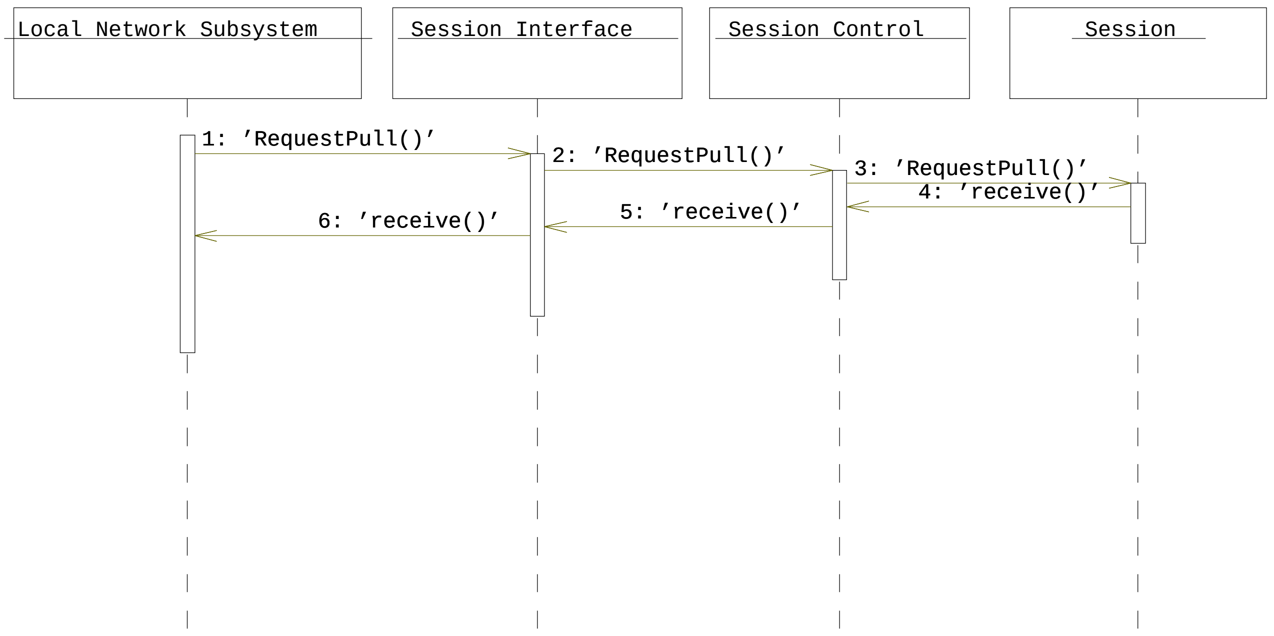
Client Side Push :



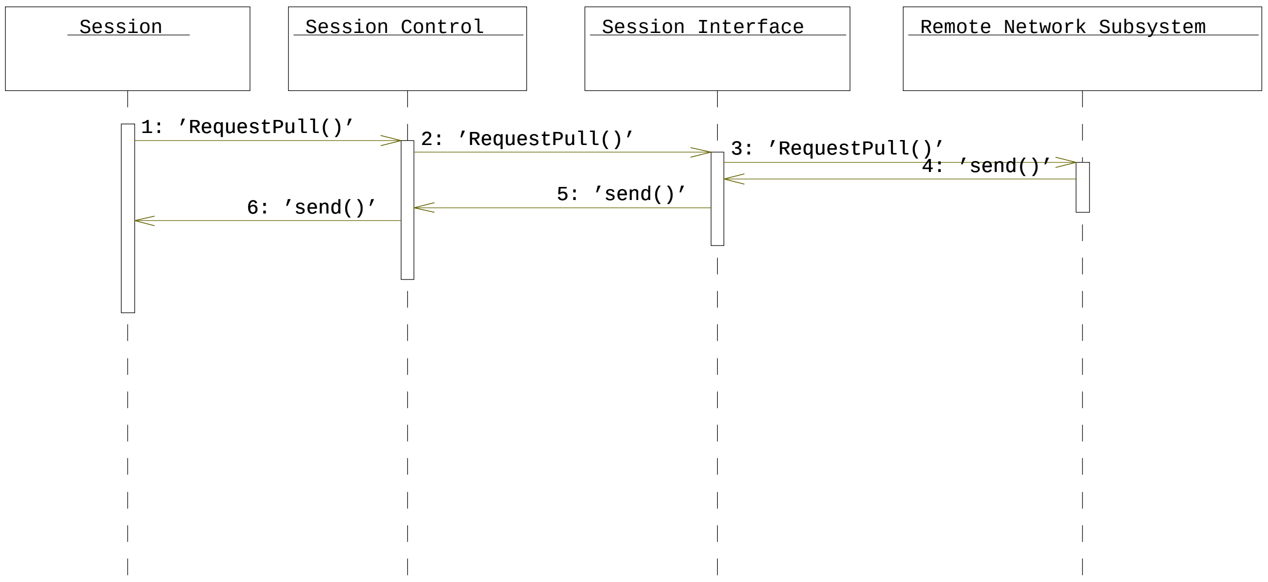
Push Delay (Client−Side / Server−Side) :



Client Side Pull :



Server Side Pull



3.4.6.1 User Interface – Navigational Paths and Screen Mockups

Network does not have any direct interaction with the users. Therefore, we do not have a user interface.

3.5 User Interface and STAR NETWORK Integration

3.5.1 Overview

The User Interface Subsystem will be designed to provide end−users with a user−friendly graphical interface for communication with all subsystems which may depend on user feedback. There will be mainly two distinguishable types of user interaction: Usage by staff in order to retrieve information provided by STAR NETWORK, and system maintenance.

Regular usage exclusively takes place on the STAR NETWORK client. All interaction will therefore be fully integrated into the existing client interface. The look&feel of all UI components will be consistent with the STAR NETWORK Visual Style Guide, based on the widgets provided by Java's AWT and the STAR NETWORK standard GUI Components.

Direct server interaction is not meant to be within the scope of end users, but administrative staff. Deeper integration into the SN client is for performance and flexibility reasons not desireable. Instead, there will be a simple standalone interface for easy handling of basic administrative tasks. Apart from retrieval and upgrade of server contents, there may be a future demand for a network−based handling of installation and upgrading of the PAID/STAR NETWORK software itself. A system to achieve this would also be integrated into the administrative interface, but will be beyond the scope of this project.

3.5.2 Functional Requirements

The User Interface Subsystem will provide a set of user−friendly interfaces which will handle display and input of data in the process of interaction between the STAR NETWORK system and end users. First of all it will provide an interface for authentication of users during login and remote access process, that is a user is required to authenticate themselves providing a valid user name and password. It will be also provided interfaces to display error messages in cases of inconsistency like unavailability of data or down network. For selecting data and updating/downloading of selected data there will be specific interfaces; the process of downloading data to the user will be provided with a specific status interface. Furthermore, each user may have his/her own preference settings for the system; the User Interface Subsystem will request the database to store and retrieve these preferences. In order to implement these tasks, the translator module will translate between the STAR NETWORK internal query language and the PAID query language.

The modified STAR NETWORK server will also capture queries from the STAR NETWORK client.

3.5.3 Nonfunctional Requirements

3.5.3.1 User Interface and Human Factors

The system is intended to be used by the executive personal of MB sale outlets, such as clerks, salesmen or mechanics. All users are considered computer novices who need extensive online help and user guidance. Therefore, it is important that the system be user−friendly. The system will have to offer appropriate access to data for each type of user, paying attention to ease of use as well as the security of sensitive information. It should always let the user know whether he or she is accessing the database on the computer or a remote database, such as MBNA. Also, when repeated access of the same data is detected, the system should notify the user of it, and − if desired − guide him or her through the process of copying the data into the local database. Since the system will be used worldwide, it is important to be able to accommodate different languages and cultures.

The system may be used on the following platforms:

- An office PC running a Java Virtual Machine with a readable display, a mouse and a keyboard.
- A laptop computer as well as a small hand−held computer for use in a garage or outside of the dealership (e.g. Palm pilot, Windows CE computers with keyboard/mouse, pen or touch screen)

The interface should be consistent across platforms. Someone normally using the system on a handheld computer should not be confused or misled by the user interface of the PC version. Every application should have the same look and feel.

3.5.3.2 Documentation

Documentation will have to be included in the form of an extensive online help system, multimedia or paper tutorials and reference manuals, for each type of user. The general user interface as well as the most common tasks performed should be described and in a comprehensive step−by−step method.

3.5.3.3 Hardware Consideration

The system may have to run on two different hardware platforms : PC and handheld computer(including laptop computers). The PC will have to be powerful enough to run a Java Virtual Machine at a satisfactory speed and have good display capabilities. Hard drive storage will not be required, except for storing the program itself, but the system will be able to make use of a large storage capacity by creating and caching a local database. The PC should have a connection either to the internet or the Mercedes−Benz network. The hand−held computer will need to be able to run Java and have enough storage space to hold both the program and the data that will be used while disconnected from the PC or the network. It should be able to connect to an office PC.

The following characteristics are recommended as the minimum for the proper graphical representation and operation:

- Screen resolution: 800x600
- Color depth: 8 bit

3.5.3.4 Performance Characteristics

The performance of the user interface mainly is depending on the performance of STAR NETWORK and the JVM: The user interface is integrated into STAR NETWORK which is a very large program. The user interface itself does not slow down the system that much.

3.5.3.5 Error Handling and Extreme Conditions

Errors such as data inconsistency or any extreme or unusual condition (e.g. network congestion, server unavailability or unreachability) should be detected by the system and displayed to the user. The system should always display clear error messages and be able to make suggestions to the user as to what measures should be taken in order to fix the problem.

3.5.3.6 System Interfacing

The user interface subsystem primarily communicates with the user input devices such as a keyboard, mouse, and possibly a touch−screen. Here, a method of counting the number of mouse clicks may need to be implemented in case the end−user pays per mouse click rather than a flat rate. Data will be displayed, or printed and sound may be used to inform the user. The system should be able to display information clearly, even on small screens and, in the case of the handheld computers, be easy to read outside, or in unusual environments.

3.5.3.7 Quality Issues

The user interface must be absolutely reliable and trustworthy, even across the different languages, because dealers may be charged according to the usage of the system.

3.5.4 System Modifications

A very important feature of the user interface is access through speech control which will make its use more comfortable for the user.

According to the fast development in the IT−industry there are more and more platforms with different sizes. So it is a feature to run the system on a Palm Pilot, a touchpad, augmented reality, etc. If the capability of the system changes the user interface may have to accommodate this.

New languages will have to be added as DaimlerChrysler expands and the User Interface (as well as online help and the tutorial) will have to be improved especially by the user’s feedback.

The look of the user interface depends on the JVM. So changes of JVM by new versions may lead to a different appearance.

3.5.4.1 Physical Environment

The user interface must be easily usable in a large variety of environments. The desktop PC version will be used in offices that are well lit and comfortable for the user. The portable version will have to remain usable in much more difficult environments: garages, any site away from the shop where a car has to be repaired (e.g. a customer’s garage, or roadside). It will have to undergo dirt, shock. The environment may be very noisy. Thus, to be used in such conditions, the user interface of the portable version will need to be particularly clear and user friendly. Moreover, unforeseen events may prevent the user from seeing or hearing a specific signal from the computer. Important information should be conveyed using several kinds of signals (e.g. a visual requester, or a sound to signal an error). The emphasis should be on displaying as much useful information as possible while remaining easily readable.

3.5.4.2 Security Issues

3.5.4.3 Resource Issues

Security and resource considerations are a general part of the PAID system as a whole, hence not directly within the scope of the UI implementation.

3.5.5 Constraints

Currently, complex software packages are soon experiencing performance problems even on fast hardware equipment when implemented on a "100% pure Java" basis. This also holds true for current versions of STAR NETWORK and the Java VM on most platforms. Deep integration of both client and server components into STAR NETWORK will therefore let the performance of the PAID subsystem be highly dependent on the performance of STAR NETWORK itself.

Sticking to the whole package being implemented in Java generally restricts its application to hardware platforms which support the Java Environment. This will − currently − not allow STAR NETWORK and PAID to be properly implemented on many common portable clients (e.g. Palm Pilot or Windows CE systems) beyond laptop computers or fully equipped handheld PCs, due to unavailability of an implementation of the Java environment on the one hand and stronger performance restrictions on smaller machines on the other hand. This situation may change over time.

3.5.6 System Model

3.5.6.1 Scenarios

Data outdated

A dealer requests data from STAR NETWORK. He knows he has got the data because he downloaded it a few months ago for another customer. But the data he requests is no longer up to date (expired). He is now asked, if he wants to update his database or if he wants to use his outdated version and delay the update for later (because normally the information does not change a lot. He decides to download later, because the customer wants the information now. STAR NETWORK displays the data an adds to it a mark "OUTDATED". Example

ON−OFFLINE

A user is working online and the status indicator displays "ONLINE" But he decides that he does not need the network access anymore. Operating with the graphical user interface, he deactivates the online mode. The status indicator displays "OFFLINE".

Download status information

The user requests data within the STAR NETWORK application; if the data is received without errors, status information are available to the user: percent of downloading, time till downloading ends. If there are errors in the process of transmission of data or data is not available the user will get a specific error message within the status interface. The data requested by the user is returned from the local database (if available), if not the request is forwarded to the PAIDserver.

Account Info & Preferences

A dealer starts up in the morning the PAID enabled STAR NETWORK in his dealership and the system asks him whether he wants to update his local database. Because they update would take long time and he has to serve his customers, he decides to delay the update. Therefore he edits his account information containing his preferences too, and switches from automatic to manually initiated updates. In the afternoon he initiates an update, because the customers became fewer and he knows that at this time the data transfer is fast enough. After customizing his local database in the preferences, only the data according to his business requirements are updated.

Data−request

The user requests data from the STAR NETWORK Server. The STAR NETWORK Server returns the requested data if they are locally available. If the data are not available locally the STAR NETWORK Server asks the PAID Server for this data and returns them to the user if the transmission takes place and the requested data are available at the PAID database, otherwise he returns an error message.

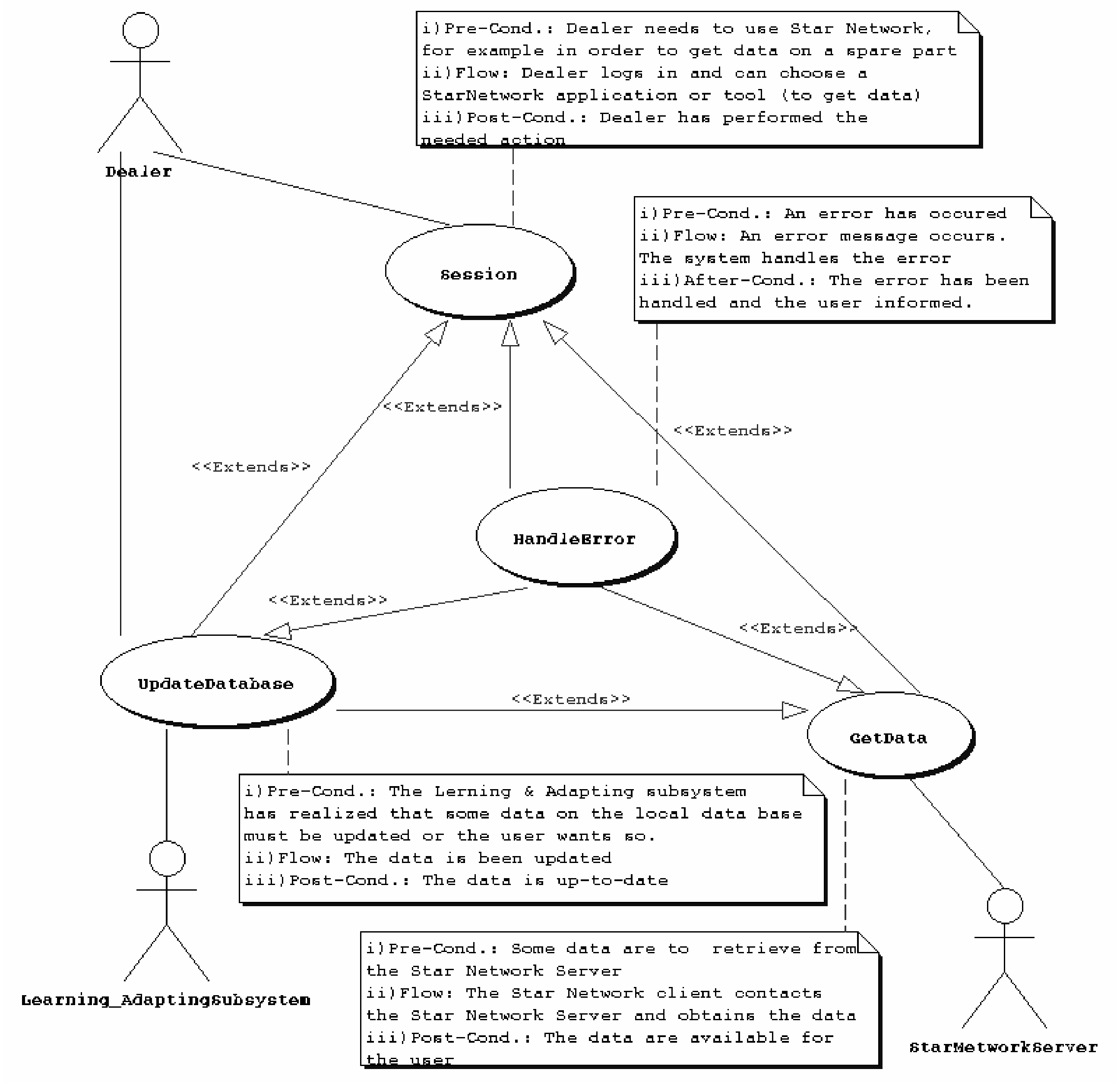
Data not available

A dealer wants to make the data in his local system up to date. He invokes the accordant operation through the graphical user interface or accepts the question of the system if he wants to bring the local database up to date. But the connection to the central database cannot be established or the new data on the central data base is not available. A message "Data not available. Pleas try later!" appears.

Working session timed out

A dealer works a while normally with the system. If his session idles a certain amount of time, a newly initiated session must re−authenticated.

3.5.6.2 Use Case Models


i) Pre-Cond.: Dealer needs to use Star Network, for example in order to get data on a spare part
ii) Flow: Dealer logs in and can choose a StarNetwork application or tool (to get data)
iii) Post-Cond.: Dealer has performed the needed action
Dealer
Session
i) Pre-Cond.: An error has occured
ii) Flow: An error message occurs. The system handles the error
iii) After-Cond.: The error has been handled and the user informed.
<<Extends>>
<<Extends>>
<<Extends>>
HandleError
<<Extends>>
<<Extends>>
UpdateDatabase
<<Extends>>
GetData
i) Pre-Cond.: The Lerning & Adapting subsystem has realized that some data on the local data base must be updated or the user wants so.
ii) Flow: The data is been updated
iii) Post-Cond.: The data is up-to-date
i) Pre-Cond.: Some data are to retrieve from the Star Network Server
ii) Flow: The Star Network client contacts the Star Network Server and obtains the data
iii) Post-Cond.: The data are available for the user
Learning_AdaptingSubsystem
StarNetworkServer

Actors

The actors of the above Use Case Diagram are the following:

- Learning and Adaption Subsystem
- STAR NETWORK Server
- Dealer

For more datailed information see the diagram above.

Use Cases

The Use Cases are:

- Session
- Update Database
- Get Data
- Handle Error

For more datailed information see the diagram above.

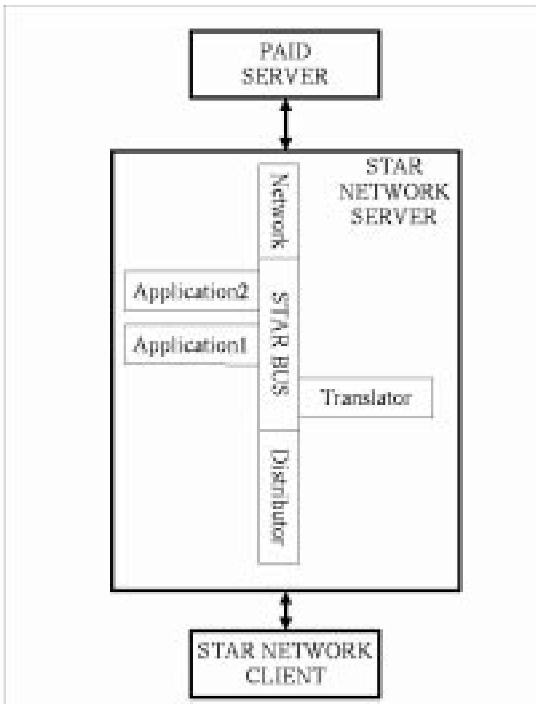
3.5.6.3 Object Models

We are recommending the following system design. This design is just a preliminary design and is likely to change in the future. If this design is realizable depends on information about STAR NETWORK that is not available at the moment.

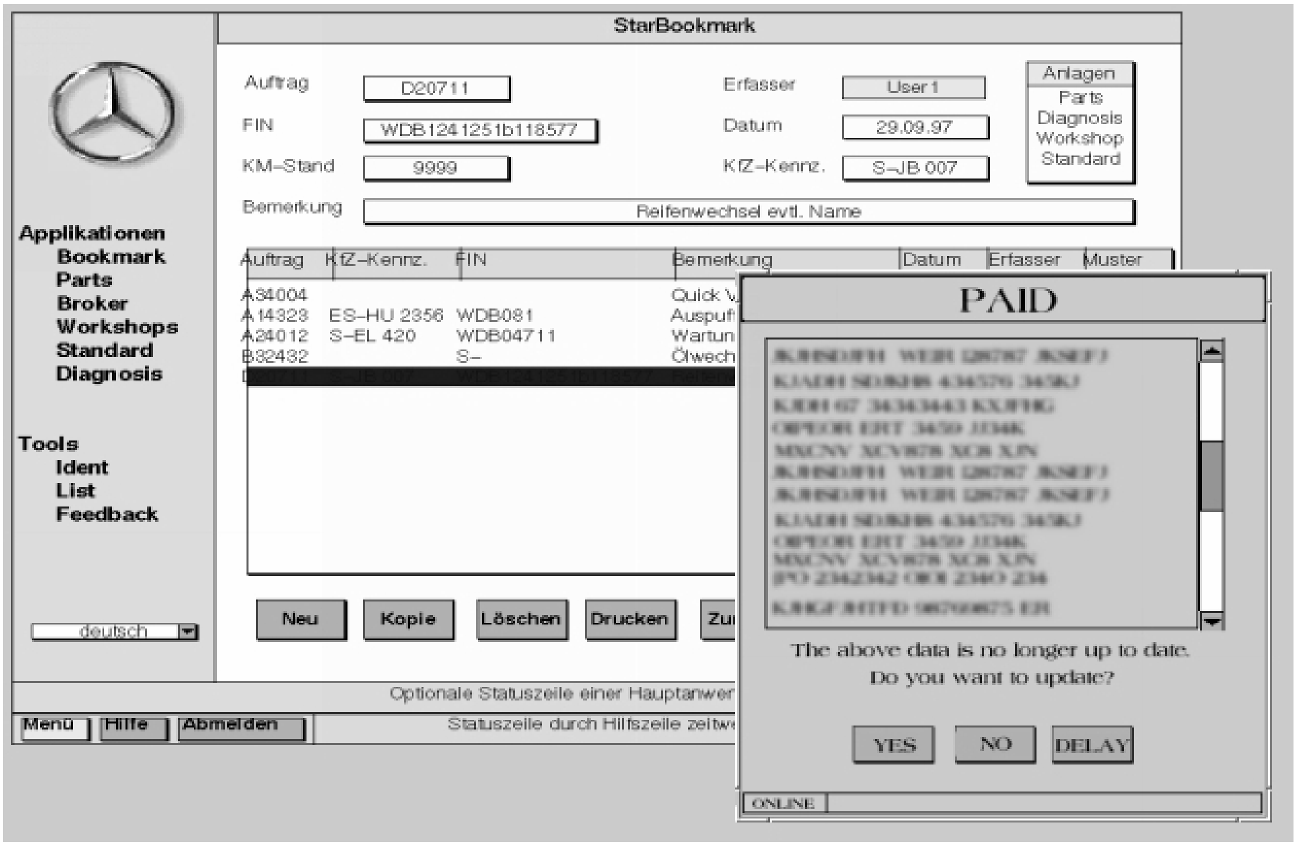
Overview of the intended STAR NETWORK integration:

All requests from the STAR NETWORK client are intercepted by the Distributor. This module sends the original request to the Translator module where it is translated into the native PAID query language. The query is then send to the PAID SERVER which tries to get the required information and send it back to the STAR NETWORK CLIENT.

There again the data is send to the translator module that produces data in the original STAR NETWORK language. By use of the STAR BUS the data goes to the required target application.



3.5.6.4 User Interface – Navigational Paths and Screen Mockups



StarBookmark

Auftrag	D20711	Erfasser	User 1
FIN	WDB1241251b118577	Datum	29.09.97
KM-Stand	9999	KfZ-Kennz.	S-JB 007
Bemerkung	Reifenwechsel evtl. Name		

Anlagen: Parts, Diagnosis, Workshop, Standard

Auftrag	KfZ-Kennz.	FIN	Bemerkung	Datum	Erfasser	Muster
A34004			Quick V			
A14323	ES-HU 2356	WDB081	Auspuf			
A24012	S-EL 420	WDB04711	Wartun			
B32432		S-	Ölwech			
D20711	S-JB 007	WDB1241251b118577	Reifen			

Optionale Statuszeile einer Hauptanwer

Statuszeile durch Hilfszeile zeitwe

PAID

[illegible]

The above data is no longer up to date.
Do you want to update?

ONLINE