Softwareprojekt 2005-2007

Software-Entwurf

Version: 1.0
Letzte Änderung: 27.04.2006 12:14

4AHD: Eichinger, Hofer, Pieringer | **4DHD:** Pointner, Schenk

Projektleiter	Peter Hofer	
Projektleitung 4DHD	Reinhard Pointner	
Verantwortlich	Peter Hofer	
Verantwortlich 4DHD	Reinhard Pointner	
Erstellt	05.10.2005 13:34	
Bearbeitungszustand		In Bearbeitung
	*	Vorgelegt
		Fertiggestellt
V-Modell-Version	XT 1.1	

Änderungsverzeichnis

Änderung			Geänderte Kapitel	Beschreibung der Änderung	Autor	Zustand
Nr	Datum	Version				
1	3.3.2006	1.0	Alle	Initiale Produkterstellung	JP	
2	29.3.2006	1.1	Alle	Kompromiss VM97/XT	PH	

Prüfverzeichnis

Datum	Geprüfte Version	Anmerkungen	Prüfer	Neuer Produktzustand

Inhaltsverzeichnis

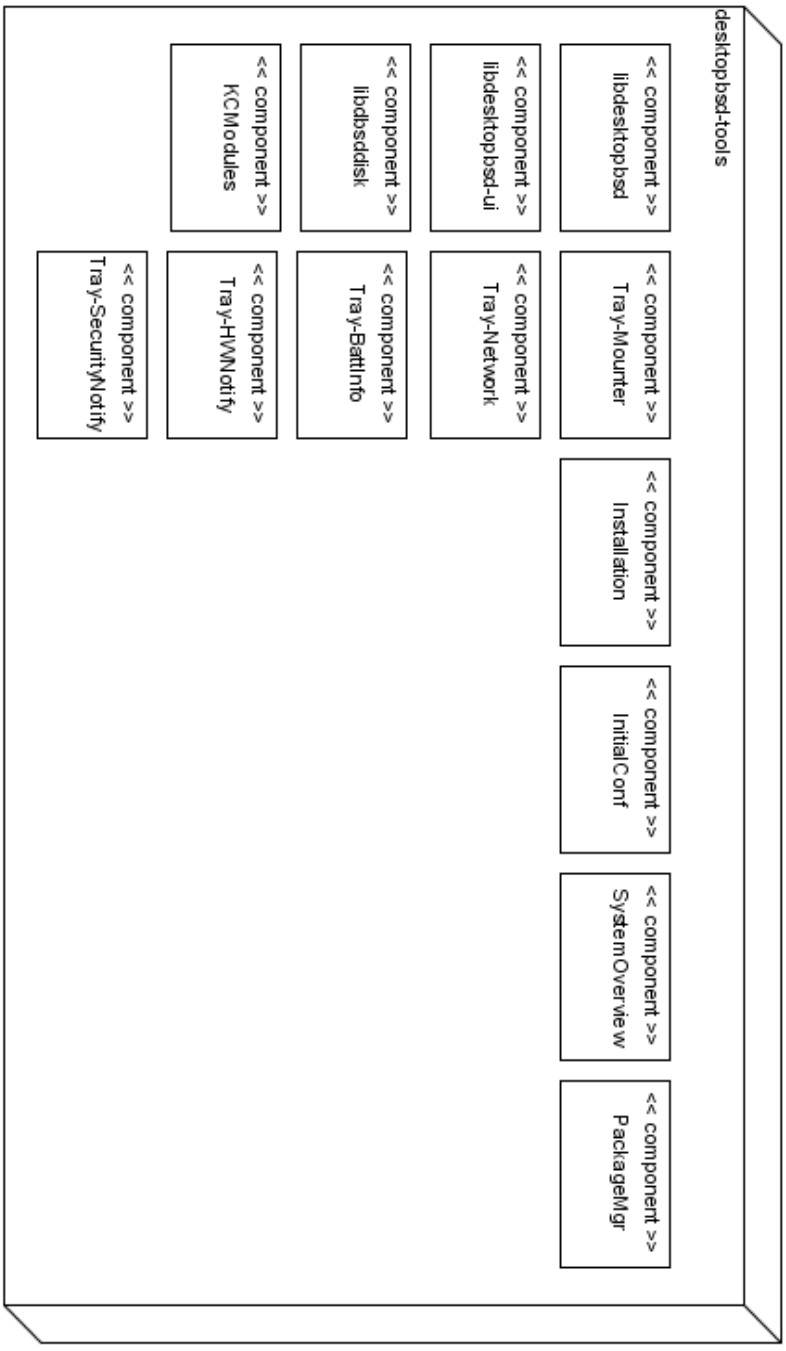
1. Einleitung

Der Software-Feinentwurf beinhaltet die Festlegung der Vorgaben und Details für die Realisierung jeder SW-Komponente, jedes SW-Moduls und jeder Datenbank, Betriebsmittel und Zeitbedarf der einzelnen Architekturelemente und der gesamten SW-Einheit.
Für jede in der Softwarearchitektur spezifizierte Softwareeinheit wird eine genauere Funktionsbeschreibung angegeben. Dazu gehört auch die Beschreibung von Anwendungsfällen und Aktivitäten. Weiters wird für jede Softwareeinheit die Umgebung angeführt, in der sie läuft bzw. die Einbettung in das Gesamtsystem oder Unterstützungssysteme. Hierzu werden die einzelnen Schnittstellen spezifiziert. Weiters wird angegeben, wie die einzelnen Softwareeinheiten realisiert werden, wie beispielsweise verwendete Klassen und Methoden. Auch das Ausnahmeverhalten ist zu berücksichtigen.

2. Architektursichten

2.1 Komponentendiagramm

KD / Komponentendiagramm



- *libdesktopbsd*

  Das „Backend“. Diese Komponente beinhaltet die Kernklassen, die die eigentliche Arbeitsaufgabe erledigen (beispielsweise die Systemzeit von einem Zeitserver aktualisieren).

- *libdesktopbsd-ui*

  Das „Frontend“. Beinhaltet alle grafischen Benutzerschnittstellen, mit denen der Endbenutzer die Funktionalität in Anspruch nimmt.

- *libdbsddisk*

  Beinhaltet primär Low-Level-Code, der für den Partitionsmanager benötigt wird.

- *KCModules*

  Enthält entsprechende Klassen, um die grafischen Benutzerschnittstellen im KDE-Kontrollzentrum einbinden zu können.

- *Tray-Mounter*

  Ein Symbol in der Kontrollleiste, mit dem Dateisysteme (beispielsweise von einem USB-Stick) eingebunden und sicher entfernt werden können.

- *Tray-Network*

  Symbol in der Kontrolleiste, über das sich die Netzwerkkonfiguration aufrufen lässt und das Netzewerk-Statusänderungen (Änderung von IP und anderen Verbindungsdaten) anzeigt

- *Tray-BattInfo*

  Symbol in der Kontrollleiste, das über die aktuelle Stromquelle und bei Akkubetrieb über den Ladestand Auskunft gibt.

- *Tray-HWNotify*

  Benachrichtigung über Hardware-Ereignisse, beispielsweise wenn ein USB Gerät eingesteckt/ausgesteckt wurde, der Computer zu heiß wird oder sich die Stromquelle ändert.

- *Tray-SecurityNotify*

  Sicherheitsupdate-Symbol in der Kontrollleiste, welches angezeigt wird, wenn Updates verfügbar sind.

- *Installation*

  Beinhaltet den grafischen Installationsassistenten für das DesktopBSD-Betriebssystem.

- *InitialConf*

  Diese Komponente beinhaltet den Wizard, der beim ersten Start angezeigt wird um das System einzurichten, z.B.: Benutzer anlegen, Systempasswort vergeben,...

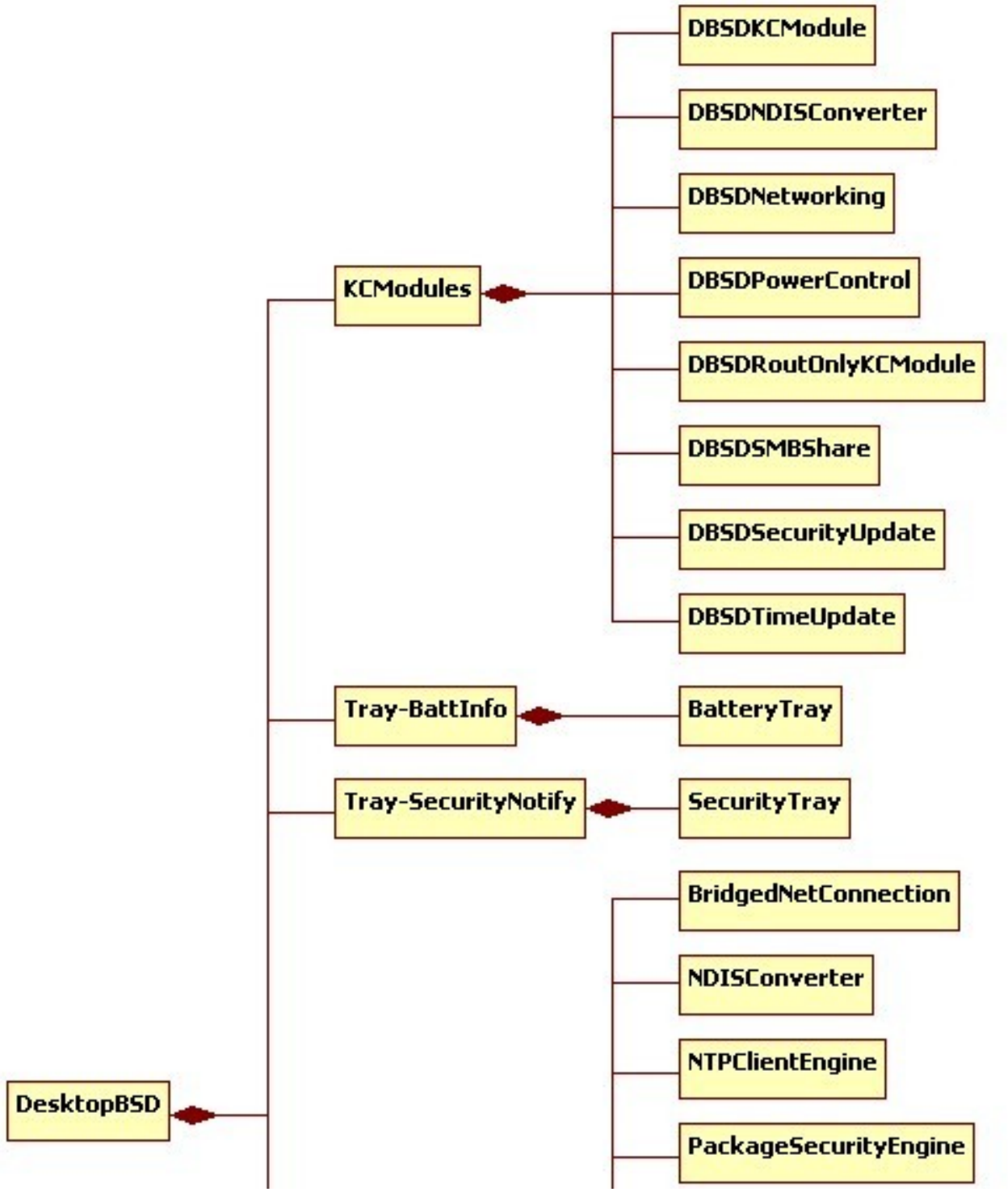
- *SystemOverview*

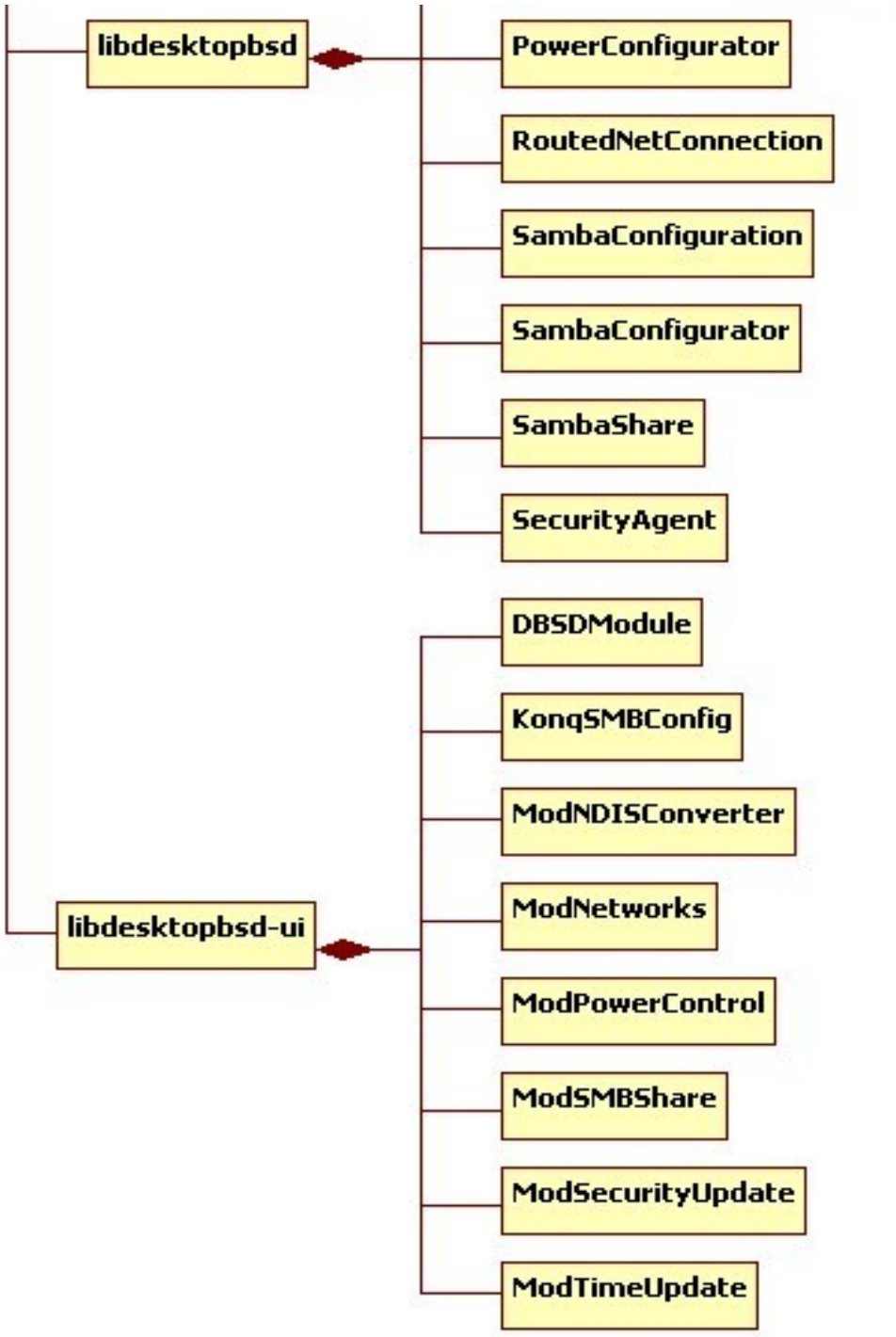
  Beinhaltet Programme zur Anzeige von Systeminformationen am Desktop.

- *PackageMgr*

  Paketmanager der für die Installation von zusätzlichen Softwarepaketen über das „Port-System“ von FreeBSD.

2.2 Aggregationshierarchie





3. Klassendiagramm

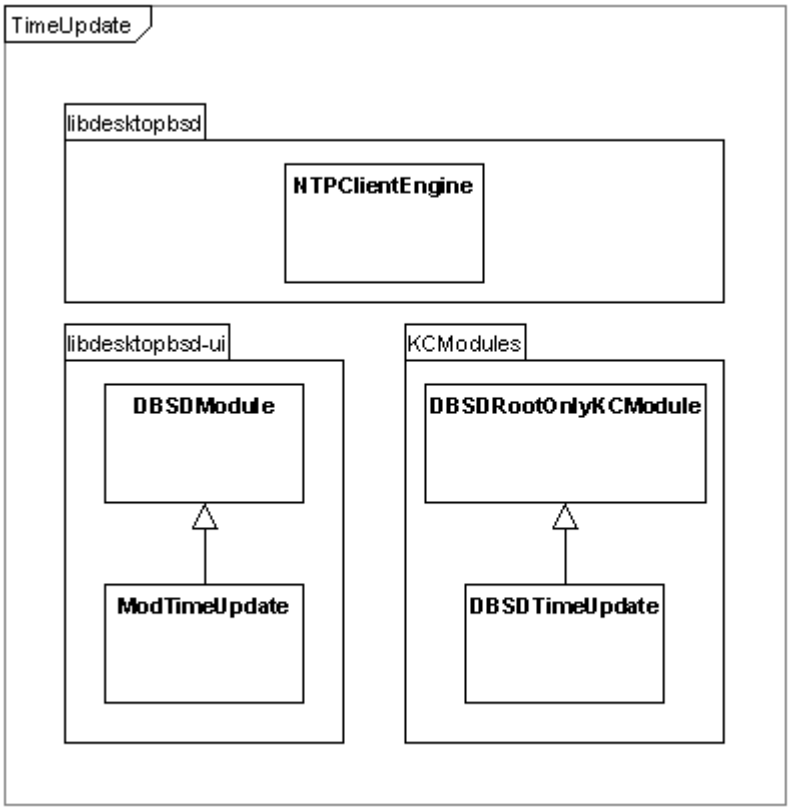
Siehe Zettel.

4. Beschreibungen

4.1 Allgemeine Strukturen

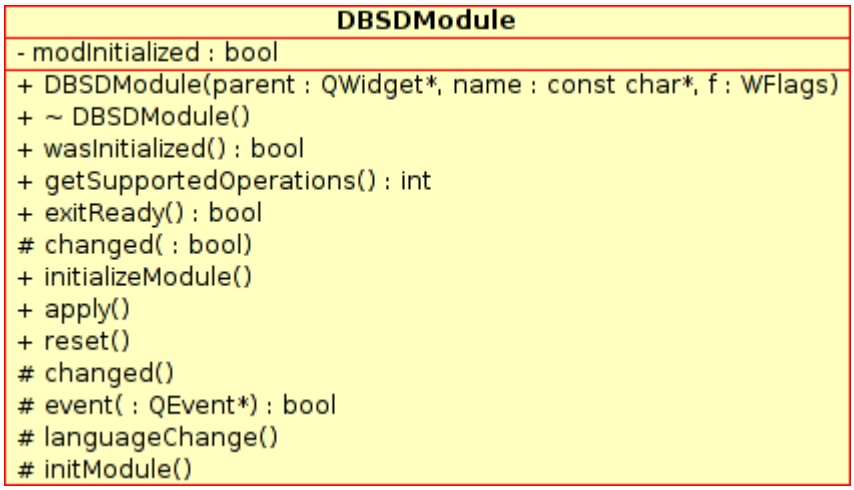
4.1.1 Ableitung von DBSDModule

CLD / TimeUpdate



Am obigen Beispiel des Zeitabgleichsmoduls wird deutlich:

- Die Klasse ***NTPClientEngine*** in *libdesktopbsd* ist das Backend – eine Klasse, die die eigentliche Arbeit übernimmt.
- Die Klasse ***ModTimeUpdate*** in *libdesktopbsd-ui* ist in diesem Fall die grafische Oberfläche – also das Frontend zur *NTPClientEngine*-Klasse in *libdesktopbsd.* Sie bietet dem Benutzer die Funktionalität des Backends an. Alle solchen „grafischen Konfigurationsmodule“ sind von *DBSDModule* abgeleitet:
- Die Klasse ***DBSDModule*** ist eine Abstraktion aller grafischen Konfigurationsmodule und beinhaltet einen Mechanismus zum Initialisieren des Objekts und Methoden zum Speichern, Laden und Zurücksetzen der Einstellungen, die von abgeleiteten Klassen implementiert und verwendet werden müssen:



- ***KCModule*** (nicht abgebildet) ist eine Klasse von *KDE*. Diese Klasse erfüllt im Prinzip den selben Zweck wie *DBSDModule*. Das Implementieren von KCModule ist notwendig, um ein Konfigurationsmodul im Kontrollzentrum anzeigen lassen zu können. Daher gibt es zwei von *KCModule* abgeleitete Wrapper-Klassen, über die *DBSDModule*-Objekte angesteuert werden können (diese Klassen delegieren Methoden weiter an ein DBSDModule-Objekt):
- ***DBSDKCModule*** delegiert nur einfach an ein angegebenes DBSDModule-Objekt.
- ***DBSDRootOnlyKCModule*** besitzt einen Mechanismus, um vor dem Anlegen eines DBSDModule-Objekts zu prüfen, ob der ausführende Benutzer im Systemverwaltungsmodus ist. Ist er das nicht, muss er das Systempasswort eingeben, und das Konfigurationsmodul wird in einem eigenen Fenster geöffnet.
- *DBSDTimeUpdate* verwendet diesen Mechanismus von DBSDRootOnlyKCModule und muss außer einem Konstruktor nichts implementieren.

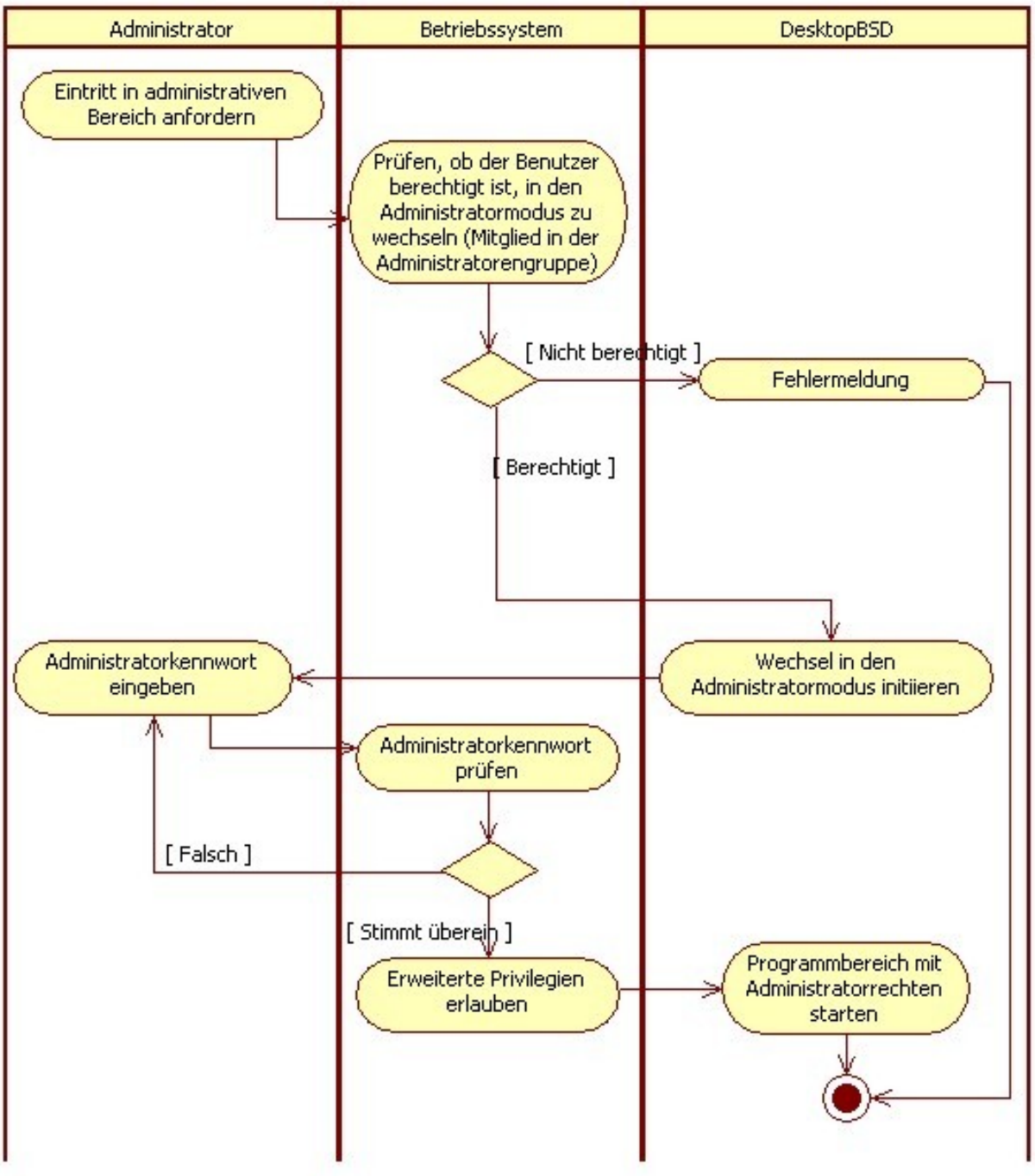
4.1.2 Speicherfreigabe durch den Aufrufer

Wird von einer Methode ein Objekt angelegt und ein Zeiger darauf zurückgegeben, ist allgemein der Aufrufer dafür verantwortlich, dass der Speicher nach der Benutzung freigegeben wird.

4.1.3 Allgemeine Aktivitätendiagramme

4.1.3.1 Eintritt in administrativen Bereich

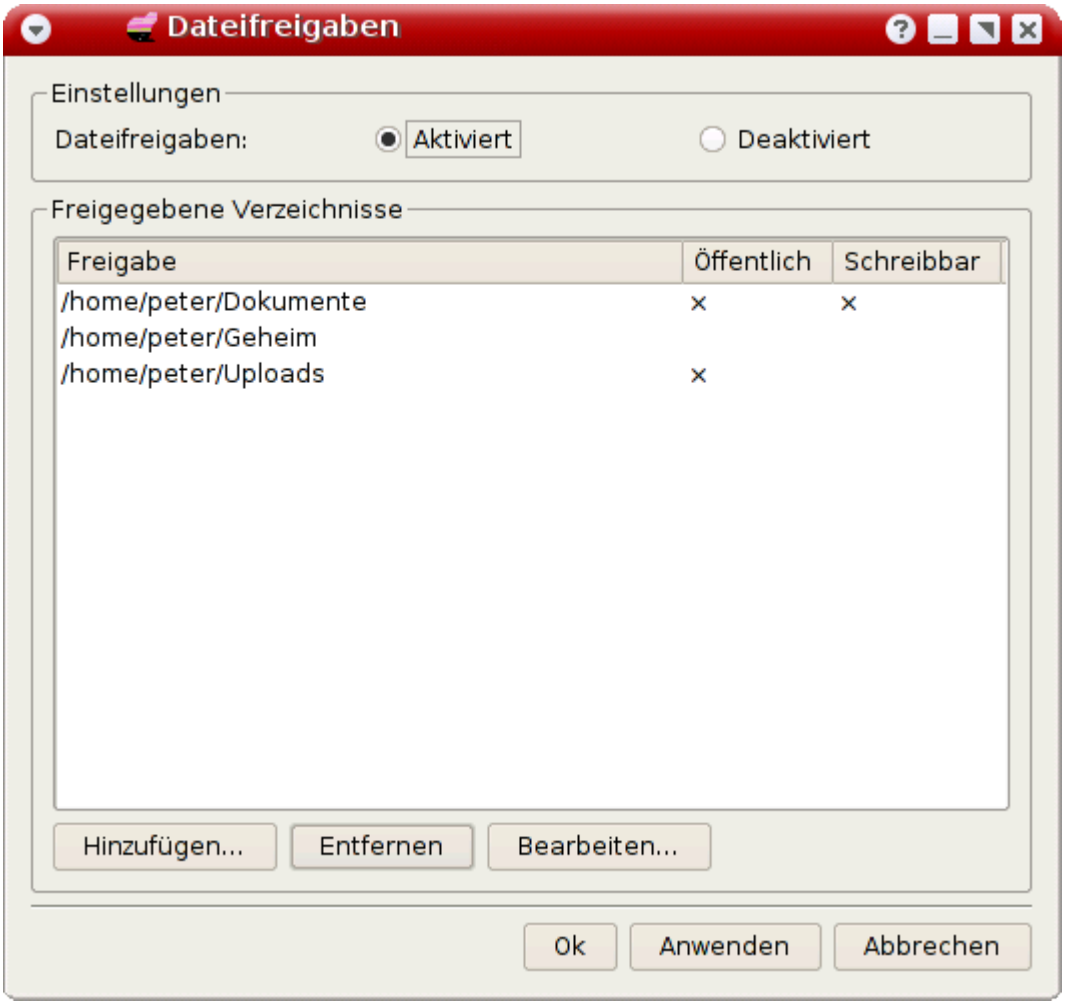
ACD / Administrationsmodus

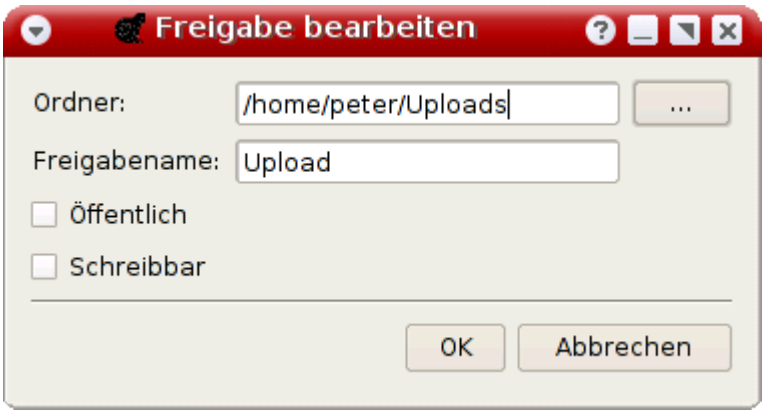


Dieses ACD beschreibt den Wechsel eines Benutzers in den Systemverwaltungsmodus, um Vorgänge auszuführen, zu denen er als gewöhnlicher Benutzer keine Berechtigung hat.

4.2 Grafische Konfiguration von Dateifreigaben

4.2.1 GUI-Prototypen





4.2.2 Funktion

(Siehe Klassendiagramm)

- SambaConfigurator enthält Methoden zum Hinzufügen, Entfernen und Bearbeiten von Dateifreigaben. Die Dateifreigabe selbst kann auch aktiviert und deaktiviert werden.
- KonqSmbConfig ist eine Erweiterung für den KDE-Dateimanager *Konqueror.*

Öffnet der Benutzer in Konqueror den Eigenschaftendialog eines Ordners, werden ihm zusätzliche Optionen zur Freigabe des Ordners angeboten.

Funktionsübsicht

- Dateifreigabe deaktivieren
- Dateifreigabe aktivieren
- Dateifreigabe erstellen
- Dateifreigabe bearbeiten
- Dateifreigabe löschen

4.2.3 Umgebung

Die grafische Dateifreigabe wird im Konqueror als TabPage in den Eigenschaften einer Datei/Ordners realisiert.

4.2.4 Schnittstellen

Funktion: Dateifreigabe über SMB-Protokoll

- Änderung der Konfigurationsdatei *smb.conf*
  - Einzelne Freigabe:

```
[Freigabename]
path = /verzeichnispfad
```

  - Servername

```
[global]
...
server string = Servername
```

  - Arbeitsgruppe

```
[global]
...
workgroup = Arbeitsgruppe
```

4.2.5 Realisierung

Als Programmiersprache wird C++ und das Qt-Framework verwendet.

Aus den KDE-Bibliotheken werden folgende Elemente verwendet:

- KSystemTray

4.2.5.1 Methoden

SambaShare:

- *int setReadOnly(bool ro)*

  Setzt für eine Dateifreigabe die Rechte, d.h. ob Dateien nur gelesen werden dürfen oder ob sie auch angelegt, verändert oder gelöscht werden dürfen.

- *void addGuestUser(const QString& username)*

  Fügt einen User in die Liste der „Guests“. Diese Liste beinhaltet diejenigen Benutzer, denen es erlaubt ist eine Datei zu laden, wenn „GuestAccessAllowed“ aktiviert ist.

- *int setGuestAccessAllowed(bool guestAccess)*

  Setzt einer Dateifreigabe die Eigenschaft, dass auch Benutzer ohne eigenen Benutzeraccount zugreifen können.

- *int setPublicAccessAllowed(bool publicAccess)*

  Setzt einer Dateifreigabe die Eigenschaft, dass sie öffentlich sichtbar ist.

- *bool getReadOnly()*

  Gibt zurück, ob Dateien nur gelesen werden dürfen oder ob sie auch angelegt, verändert oder gelöscht werden dürfen.

- *bool getGuestAccessAllowed()*

  Gibt zurück, ob auch Benutzer ohne eigenen Benutzeraccount zugreifen können.

- *bool getPublicAccessAllowed()*

  Gibt zurück, ob die Dateifreigabe öffentlich sichtbar ist.

- *QPtrList<QString> getGuestUsers()*

  Gibt alle Gast-User zurück, die darauf zugreifen können, falls „*GuestAccessAllowed*“ aktiviert ist.

SambaConfiguration:

- *int addShare(const SambaShare& share)*

  Legt eine neue Dateifreigabe an.

- *void removeShare(const SambaShare& share)*

  Löscht eine bestehende Dateifreigabe.

- *QptrList<SambaShare> getShares()*

  Gibt eine Liste aller Dateifreigaben zurück.

- *SambaShare* getShare(const QString& path)*

  Gibt die Dateifreigabe mit dem angegebenen Pfad zurück oder NULL, wenn keine

solche existiert.

SambaConfigurator:

- *int addShare(const SambaShare& share)*

  Legt eine neue Dateifreigabe an.
- *void removeShare(const SambaShare& share)*

  Löscht eine bestehende Dateifreigabe.
- *QPtrList<SambaShare> getShares()*

  Gibt eine Liste aller Dateifreigaben zurück.
- *void startStopSamba(bool start)*

  Startet (wenn start „TRUE“ ist) bzw. beendet (wenn start „FALSE“) den Samba-Dienst.
- *SambaConfigurator* getInstance()*

  Da die Klasse SambaConfigurator als Singleton realisiert wird, gibt es nur eine Instanz der Klasse. Diese Instanz wird mithilfe dieser Methode angesprochen, d.h. „getInstance()“ gibt die einzige Instanz der Klasse SambaConfigurator zurück.

4.2.6 Lokale Daten

SambaConfiguration

- protected:
  - QPtrList<*SambaShare*> shares; // Die Liste aller Freigaben

SambaConfigurator

- *privat:*
  - SambaConfigurator instance; // Die einzige Instanz der Klasse

SambaShare

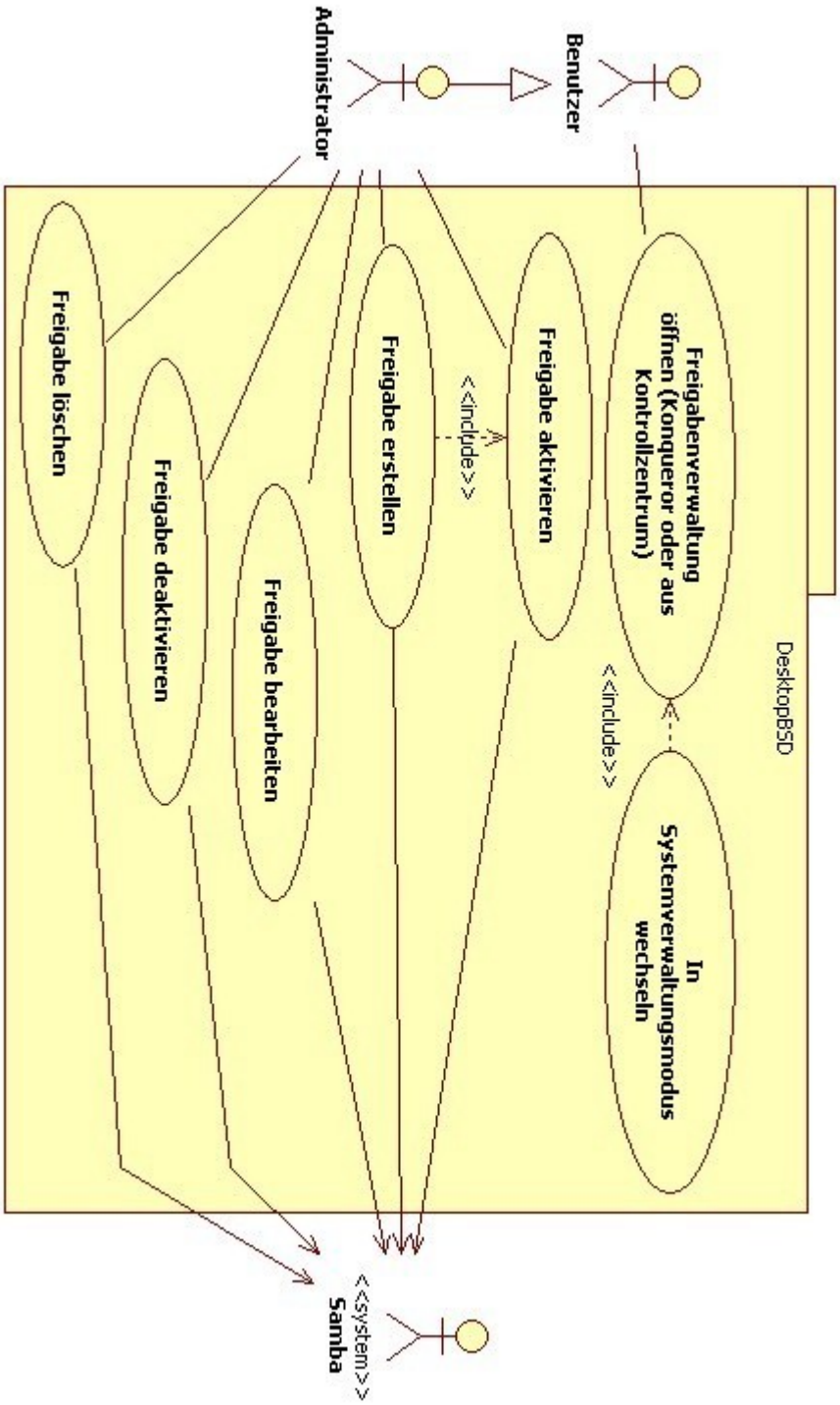
- private:
  - bool readOnly; // Ob die Freigabe beschreibbar ist.
  - bool publicAccess; // Ob die Freigabe öffentlich sichtbar ist.
  - bool guestAccess; // Ob Gäste Zugang zur Freigabe haben.
  - QString path; // Vollständiger Pfad der Freigabe
  - QStringList guests; // Liste der Gäste, die Zugang haben

4.2.7 Ausnahmeverhalten

Falls der Samba-Dienst nicht gestartet werden kann, wird ein Fehler ausgegeben.

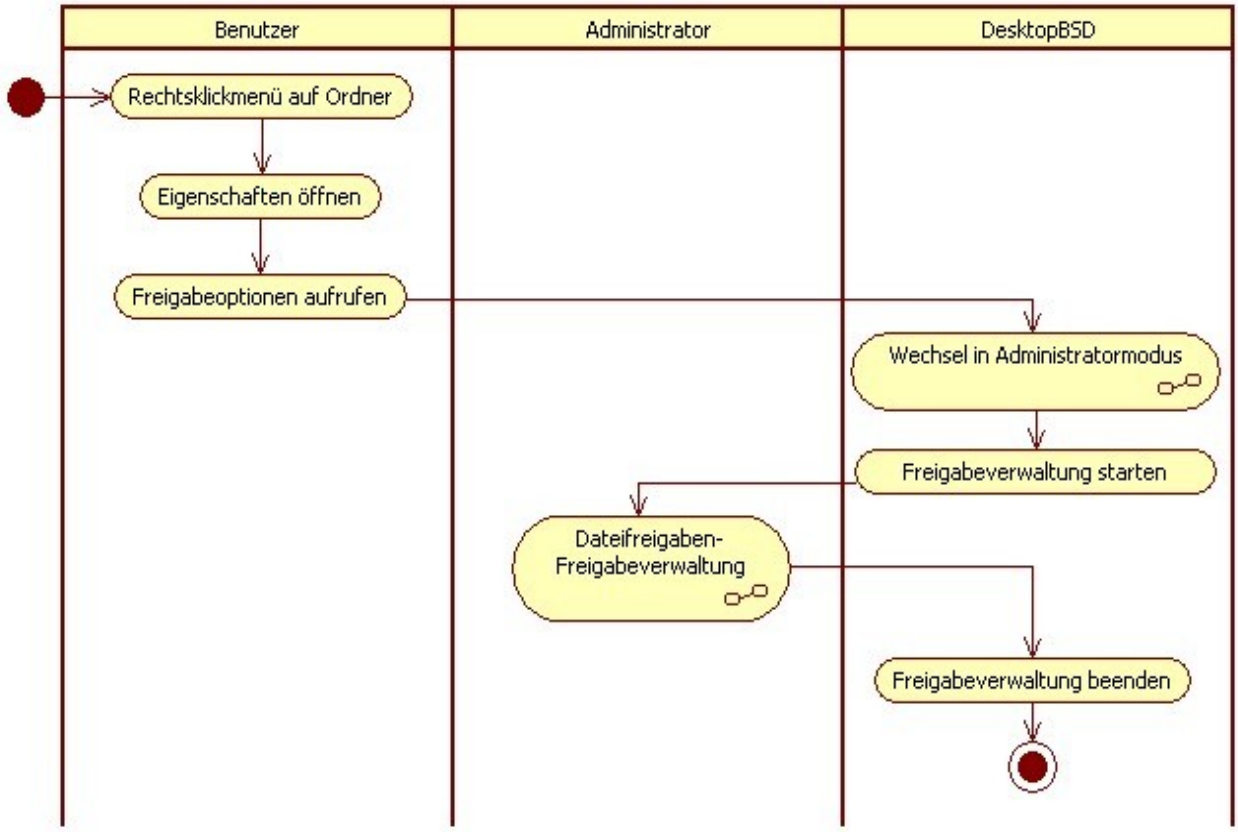
4.2.8 Anwendungsfall

UCD / Dateifreigabe

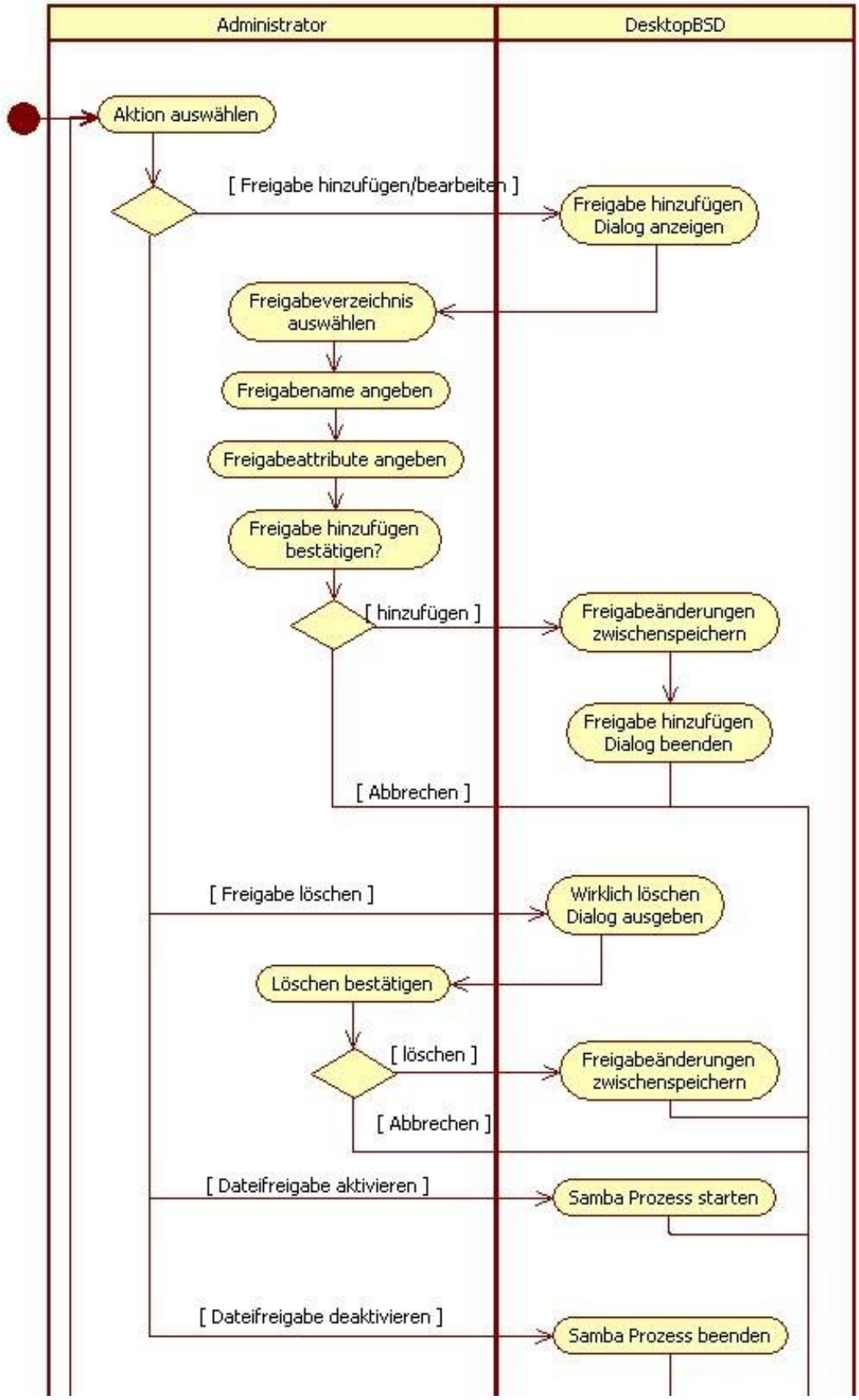


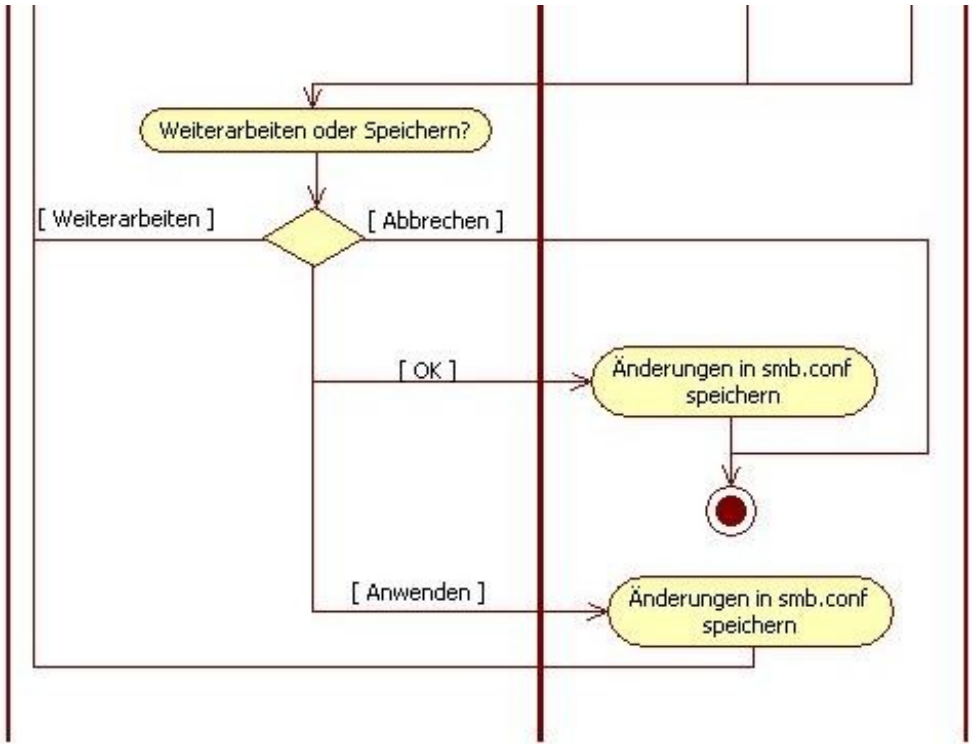
4.2.9 Aktivitätendiagramm

ACD / Dateifreigabe - Browser



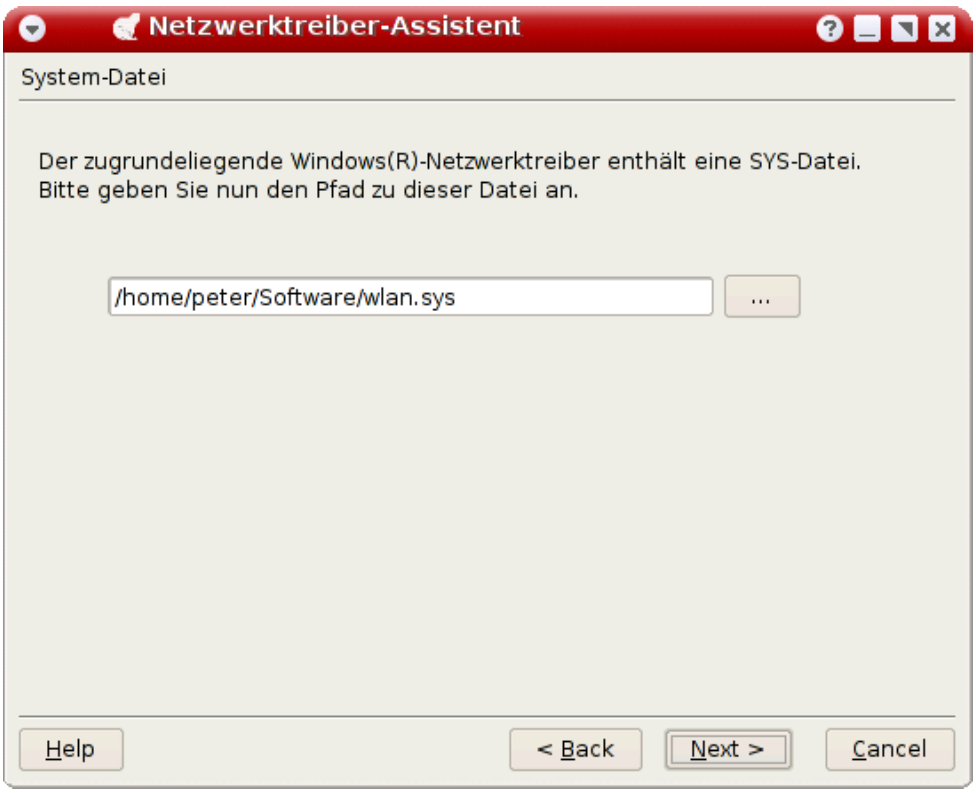
ACD / Dateifreigabe – Verwaltung




Weiterarbeiten oder Speichern?
[Weiterarbeiten]
[Abbrechen]
[OK]
Änderungen in smb.conf speichern
[Anwenden]
Änderungen in smb.conf speichern

4.3 Konvertieren von Netzwerktreibern

4.3.1 GUI-Prototypen



4.3.2 Funktion

Der Netzwerktreiber-Assistent generiert aus Windows-Netzwerktreibern FreeBSD-Netzwerktreiber, indem er die .SYS, .INF und wenn nötig, Firmwaredateien des Windowstreiber einliest und an „ndiscvt“ weitergibt. Der Assistent ist im Netzwerkmodul eingebettet.

Funktionsübersicht

- Pfad zur .sys Datei des Windowstreibers angeben
- Pfad zur .inf Datei des Windowstreibers angeben
- Pfad zu Firmeware-Files angeben (z.B.: .bin)
- FreeBSD Treiber generieren

4.3.3 Umgebung

Die Generierung von Windows Netzwerktreibern zu FreeBSD Netzwerktreibern ist in das Netzwerkmodul eingegliedert und kann einfach aufgerufen werden.

4.3.4 Schnittstellen

Funktion: Windows-Treiber zu DesktopBSD-Treiber konvertieren

- Kommandozeilenparameter:
  - *-i <inf-file>*: Treiber .INF-Datei
  - *-s <sys-file>*: Treiber .SYS-Datei
  - *-n <dev-name>*: Name, unter dem sich der Treiber später registriert
  - *-o <out-file>*: Pfad der resultierenden Headerdatei für einen Treiber
  - *-f <firmware-file>*: Angabe einer zusätzlichen Firmware

Beispiel:

```
ndiscvt -i netX500.inf -s pcx500.sys -o ndis_driver_data.h
```

Erzeugt aus der INF-Datei netX500.inf und der SYS-Datei pcx500.sys die Headerdatei ndis_driver_data.h, mit deren Hilfe ein Treiber für DesktopBSD erstellt werden kann.

4.3.5 Realisierung

Als Programmiersprache wird C++ und das Qt-Framework verwendet.

4.2.5.1 Methoden

NDISConverter

- *void generateDriver(const QString& outFile)*

  Erzeugt aus dem Windowstreiber eine Headerdatei, deren Pfad über den Parameter

festgelegt ist. Mit Hilfe dieser Headerdatei wird danach ein DesktopBSD-Treiber erstellt.

- *void setInfFile(const QString& infFile)*

  Speichert den Pfad der .INF Datei des Windowstreibers.

- *void setSysFile(const QString& sysFile)*

  Speichert den Pfad der .SYS Datei des Windowstreibers

- *void setFirmwareFile(const QString& firmwareFile)*

  Speichert Angaben einer zusätzlichen Firmware.

- *void setDeviceName(const QString& deviceName)*

  Setzt den Namen, unter dem sich der Treiber später im System registrieren soll.

- *QString getSysFile()*

  Gibt den Pfad der .SYS Datei des Windowstreibers zurück.

- *QString getInfFile()*

  Gibt den Pfad der .INF Datei des Windowstreibers zurück.

- *QString getFirmwareFile()*

  Gibt den Pfad der einer zusätzlich angegebenen Firmware zurück.

- *QString getDeviceName()*

  Gibt den Namen des Treibers zurück, unter dem sich der Treiber registrieren soll.

4.3.6 Lokale Daten

NDISConverter

- private:
  - QString devName;
  - *QptrList <QString>* firmwareFiles;
  - QString sysFile;
  - QString infFile;

4.3.7 Ausnahmeverhalten

Falls die .INF Datei im UTF-16-Format vorhanden ist, wird sie in einen ASCII-kompatiblen ASCII-Zeichensatz umgewandelt (UTF-8). Falls sie in einem unbekannten Format vorhanden ist oder nicht als .INF-Datei erkannt wird, wird der User aufgefordert, eine korrekte .INF Datei anzugeben.

Falls die .SYS-Datei keine Windows-Treiberdatei ist, wird der User aufgefordert eine korrekte .SYS-Datei anzugeben.

Falls beim Parsen der .INF-Datei und der .SYS-Datei ein Fehler passiert, wird dem User der Fehler ausgegeben und der User aufgefordert, andere Dateien auszuwählen oder den

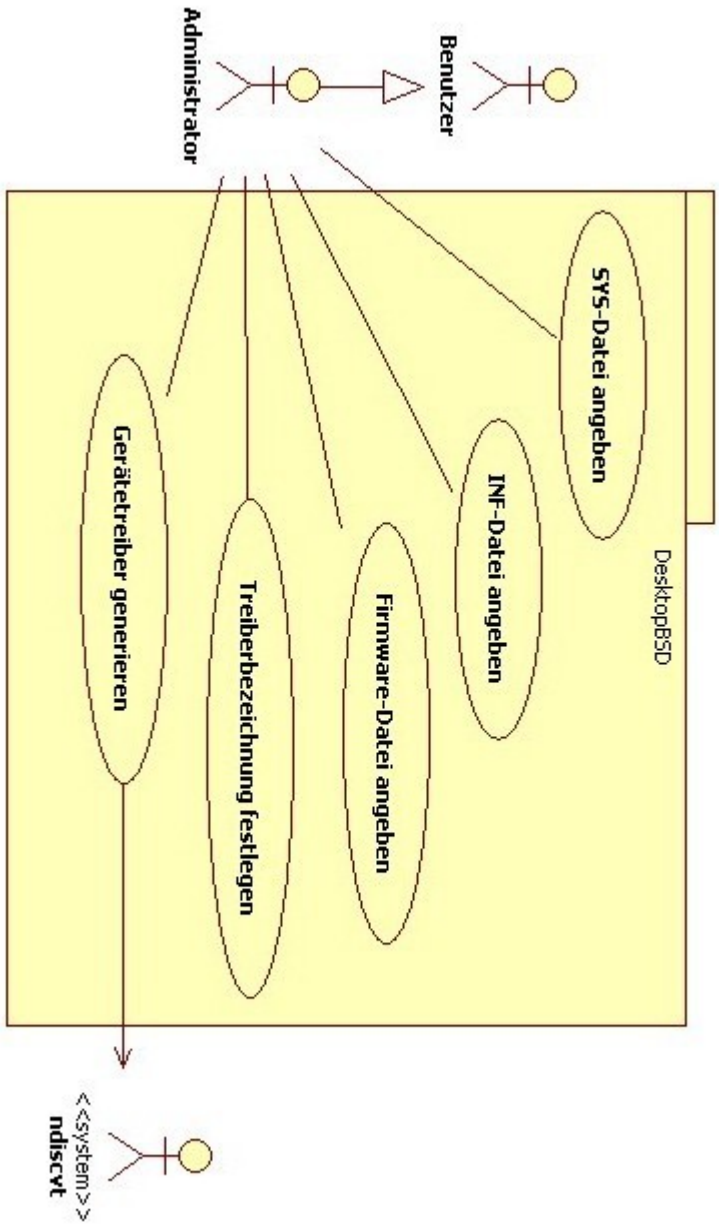
Vorgang abzubrechen.

Falls beim Hinzufügen der Firmwaredateien ein Fehler passiert, wird der User aufgefordert die Dateien zu ändern oder den Vorgang abzubrechen.

Falls das Erstellen des Treibermoduls fehlschlägt, wird der User benachrichtigt, zur End-Seite des Wizards geführt und aufgefordert, das Programm zu verlassen oder andere Dateien anzugeben.

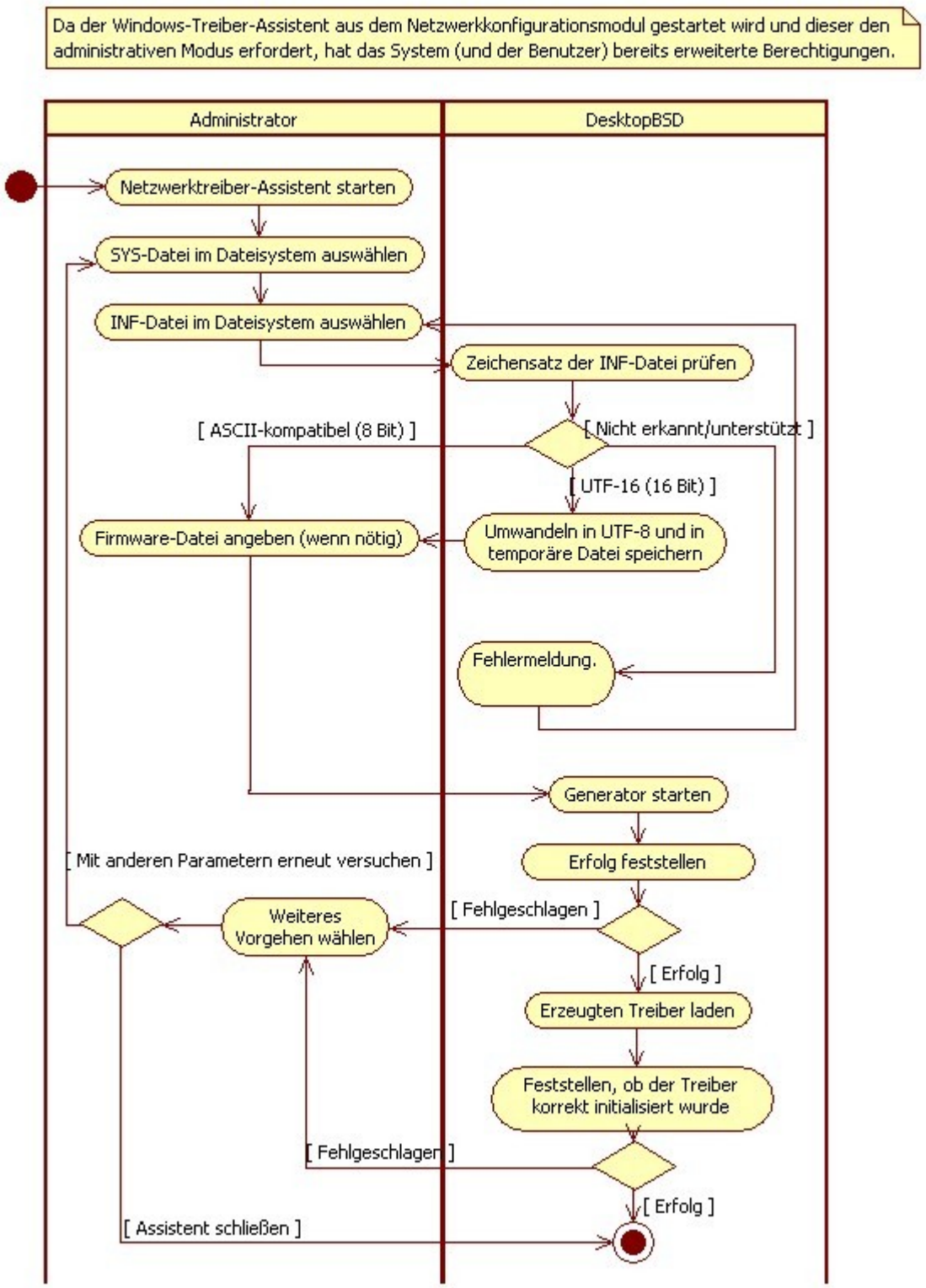
4.3.8 Anwendungsfall

UCD / NDIS-Converter



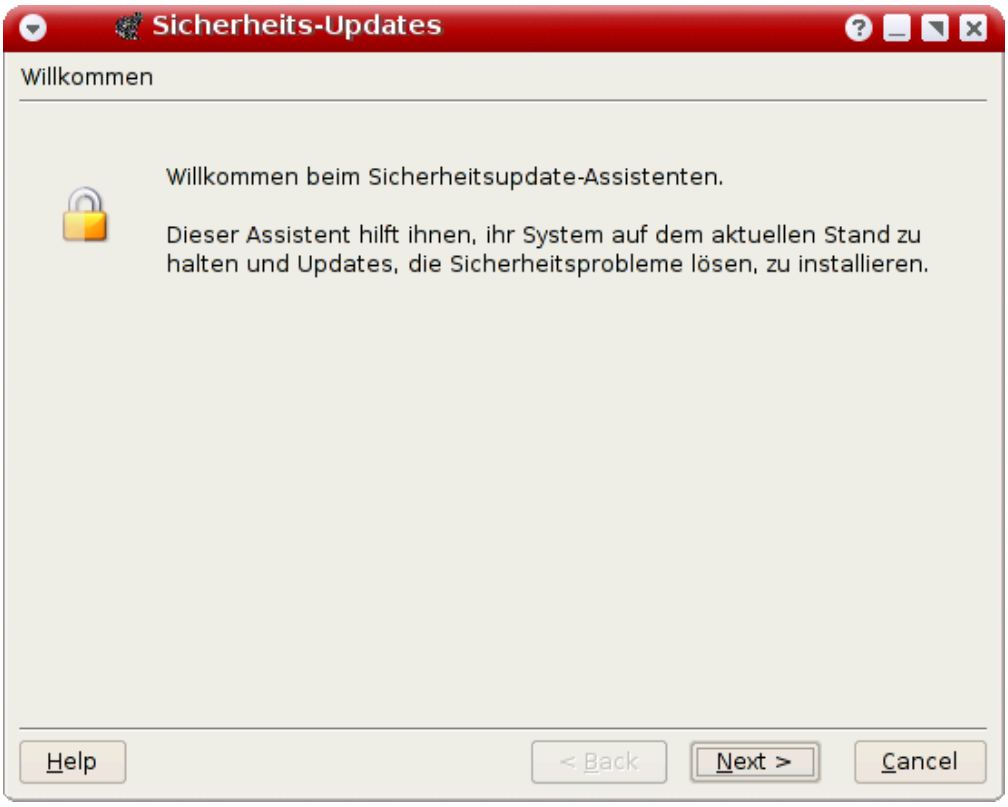
4.3.9 Aktivitätendiagramm

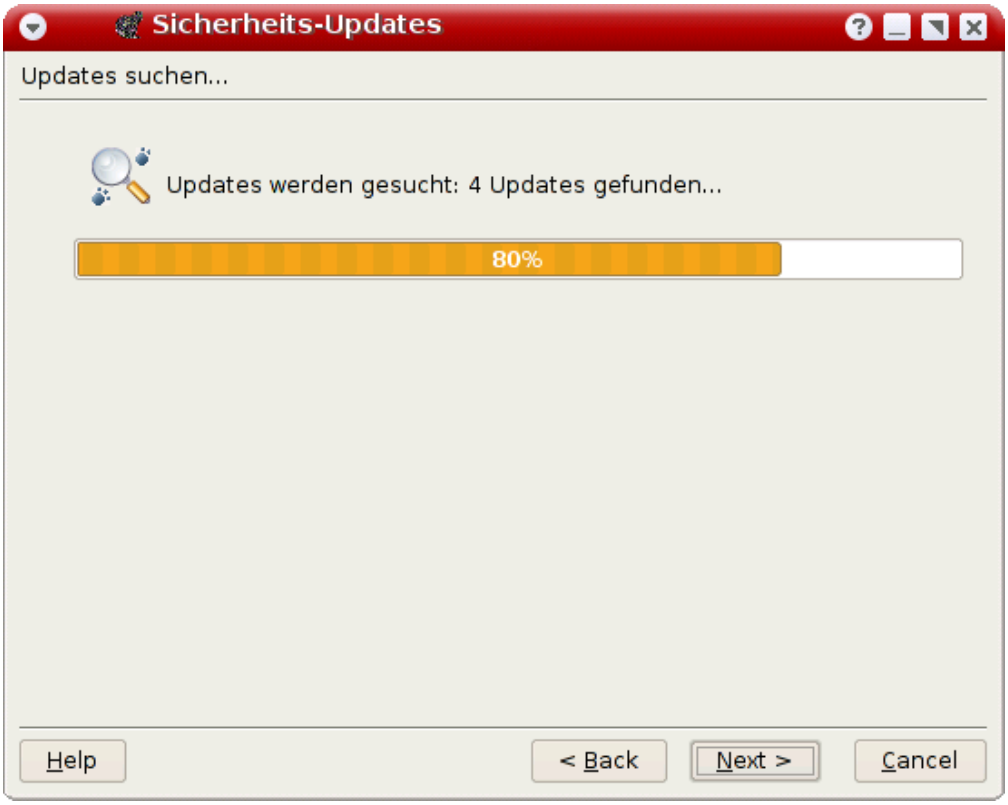
ACD / NDIS-Converter

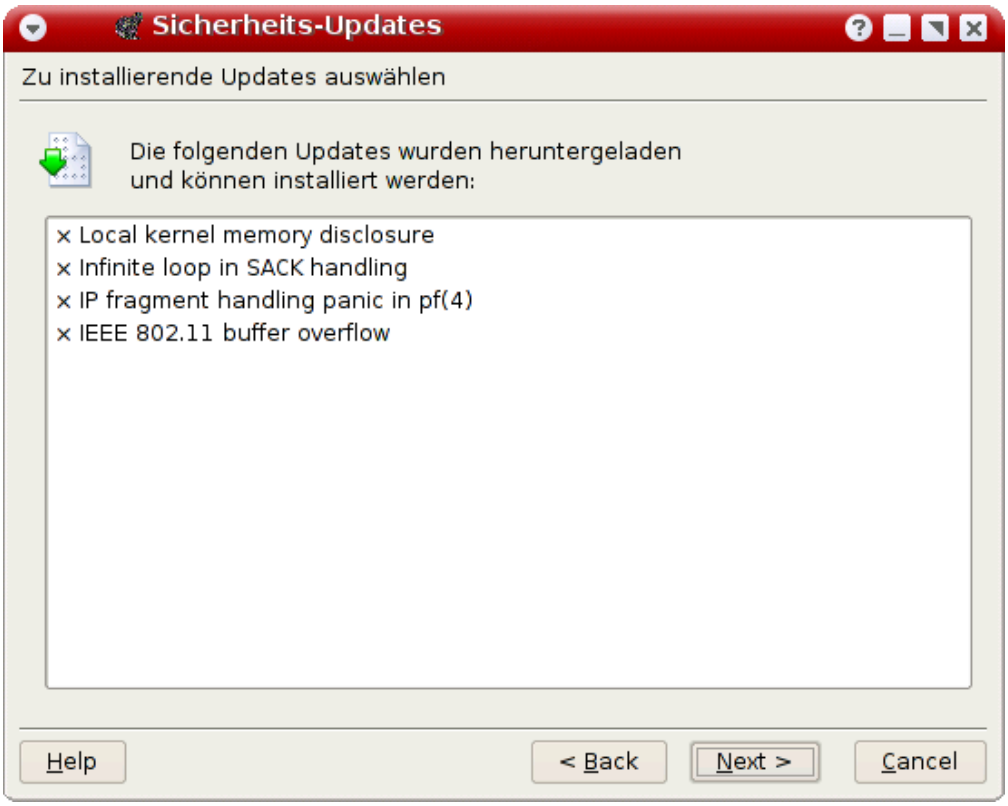


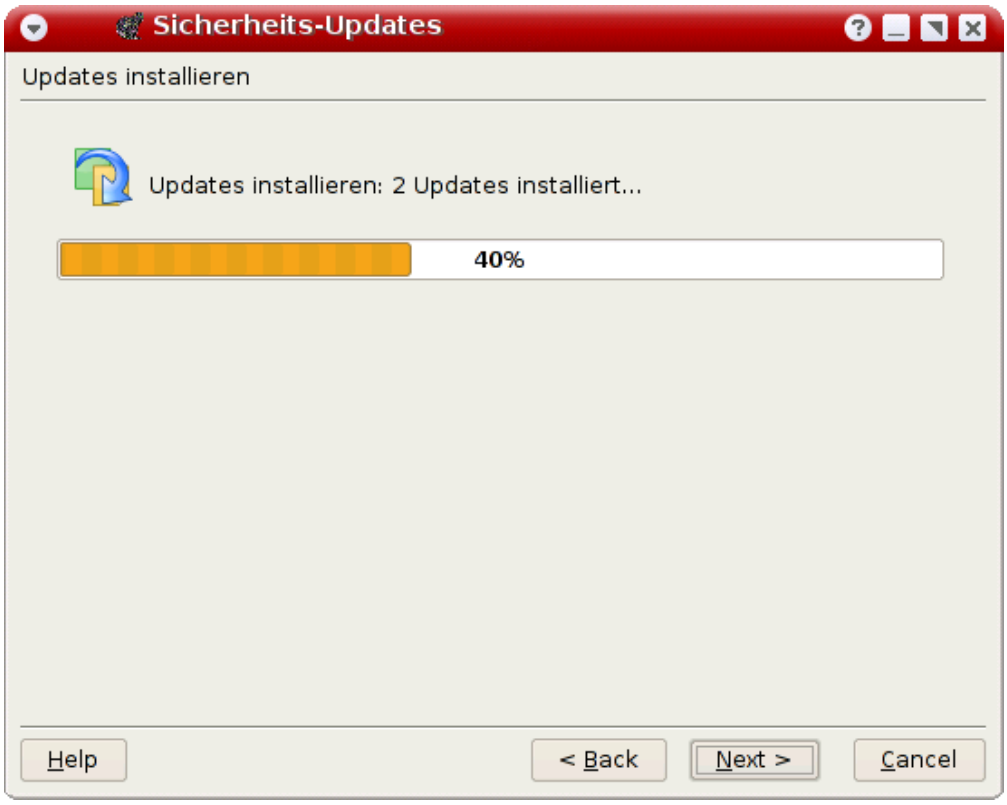
4.4 Updates am Betriebssystem

4.4.1 GUI-Prototypen




Sicherheits-Updates
Updates suchen...
Updates werden gesucht: 4 Updates gefunden...
80%
Help
< Back
Next >
Cancel


Sicherheits-Updates
Zu installierende Updates auswählen
Die folgenden Updates wurden heruntergeladen
und können installiert werden:
× Local kernel memory disclosure
× Infinite loop in SACK handling
× IP fragment handling panic in pf(4)
× IEEE 802.11 buffer overflow
Help
< Back
Next >
Cancel


Sicherheits-Updates
Updates installieren
Updates installieren: 2 Updates installiert...
40%
Help
< Back
Next >
Cancel


Sicherheits-Updates
Abgeschlossen
Die ausgewählten Updates wurden installiert. Sie sollten regelmäßig den Update-Assistenten ausführen, um vor Sicherheitsproblemen geschützt zu sein.
Help
< Back
Finish
Cancel

4.4.2 Funktion

Dieses Anwendung ist in zwei Tools aufgeteilt. Das erste Tool läuft im Hintergrund (also normalerweise nicht sichtbar) und benachrichtigt den Benutzer durch ein „KPassivePopup“ wenn Sicherheitsupdates verfügbar sind. Das zweite Programm ist dafür zuständig, die verfügbaren Updates herunterzuladen, dem Benutzer die zu installierenden Updates auswählen zu lassen und anschließend diese Updates zu installieren. Letzteres ist im Kontrollzentrum integriert und kann aber vom erstgenannten Tool aufgerufen werden, wenn Updates verfügbar sind.

(siehe Klassendiagramm)

4.2.3 Umgebung

Der Teil der Anwendung, das für das Installieren der Updates verantwortlich ist, ist im „Kontrollzentrum“ von KDE eingebettet und kann von dort aus ausgeführt werden.

Der andere Teil wird beim Systemstart mitgeladen und läuft im Hintergrund in Form eines Symbols in einer Kontrollleiste. Dabei wird in regelmäßigen Zeitabständen auf Updates geprüft und der Benutzer bei verfügbaren Updates benachrichtigt (durch ein „KpassivePopup“). Dieser kann dann direkt den Update-Installationsassistent aus dem Kontrollzentrum starten.

4.4.4 Schnittstellen

Als einzige Schnittstelle wird hier das Systemprogramm „freebsd-update“ verwendet.

Dabei werden folgende Kommandozeilenparameter verwendet:

- *fetch*: Auf Updates prüfen und diese herunterladen
- *install:* Heruntergeladene Updates verifizieren und installieren

Beispiel:

```
freebsd-update fetch

freebsd-update install
```

4.4.5 Realisierung

Als Programmiersprache wird C++ und das Qt-Framework verwendet.

Aus den KDE-Bibliotheken werden folgende Elemente verwendet:

- KSystemTray

Die Anwendung wird, wie oben beschrieben, in zwei Tools realisiert. Da das erste dieser beiden Tools hauptsächlich als Trayprogramm eingesetzt wird und nur auf Updates prüft, kommt dieses daher ohne eine grafische Benutzerschnittstelle aus. Somit reichen also die „libdesktopbsd“ Arbeitsklassen zur Programmierung dieses Tools völlig aus. Das zweite

Tool wird dann durch die Verwendung der Klasse „QProcess“ aus dem Qt-Framework von dem ersten aufgerufen.

4.4.5.1 Methoden

SecurityAgent:

- *QStringList checkForUpdates()*

  Abstrakte Methode – Abgeleitete Klassen müssen eine Liste verfügbarer Updates implementieren.

PackageSecurityAgent:

- *QStringList checkForUpdates()*

  Prüft auf verfügbare Updates für installierte Software.

- *QStringList getAffectedPackages()*

  Gibt eine Liste mit den Namen der installierten Softwarepakete zurück, für die wichtige Sicherheitsupdates verfügbar sind.

SystemSecurityAgent:

- *UnixProcess createInstallProcess()*

  Erstellt einen ausführbaren Prozess, der die entsprechenden Systemupdates installiert.

- *QStringList checkForUpdates()*

  Prüft auf verfügbare Updates am Betriebssystem.

4.4.6 Lokale Daten

-

4.4.7 Ausnahmeverhalten

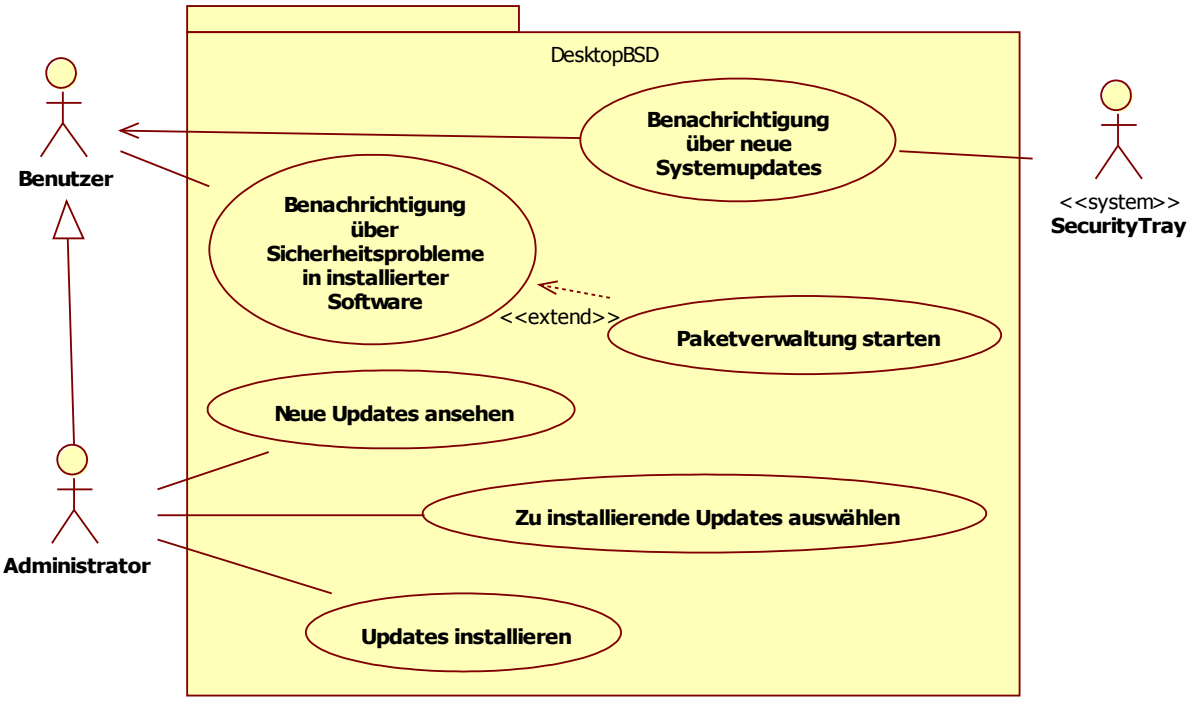
Eine Ausnahme kann zum einen auftreten, wenn keine Internetverbindung vorhanden ist oder aber auch, wenn keine sichere Verbindung zum Updateserver aufgebaut werden kann. Falls keine Internetverbindung vorhanden ist, wird der Benutzer benachrichtigt, natürlich nur, wenn der Benutzer das Tool über das Kontrollzentrum aufgerufen hat. Ansonsten (im Falle des Trayprogramms) wird dem Benutzer keine Benachrichtigung angezeigt. In diesem Fall wird einfach von Zeit zu Zeit nochmals geprüft, ob nun eine Verbindung zum Internet vorhanden ist.

Komplizierter wird es, wenn die Verbindung zum Updateserver durch ein Sicherheitsproblem nicht hergestellt werden kann. Da zur Identifikation am Updateservers ein öffentlicher Schlüssel aus einer lokalen Datei des Betriebssystem mit dem Server ausgetauscht wird könnte es hier zu einer Ausnahme kommen, wenn der Schlüssel nicht übereinstimmt. Dies ist zum Beispiel der Fall, wenn man sich einem falschen

Updateserver verbinden will oder ein „gefälschter“ Updateserver versucht, „böswillige“ Updates zu installieren. Beim Auftreten einer solchen Ausnahme wird dem Benutzer eine Benachrichtung mit der genauen Fehlerbeschreibung angezeigt und er wird weiters aufgefordert, die URL des Updateservers nochmals zu überprüfen.

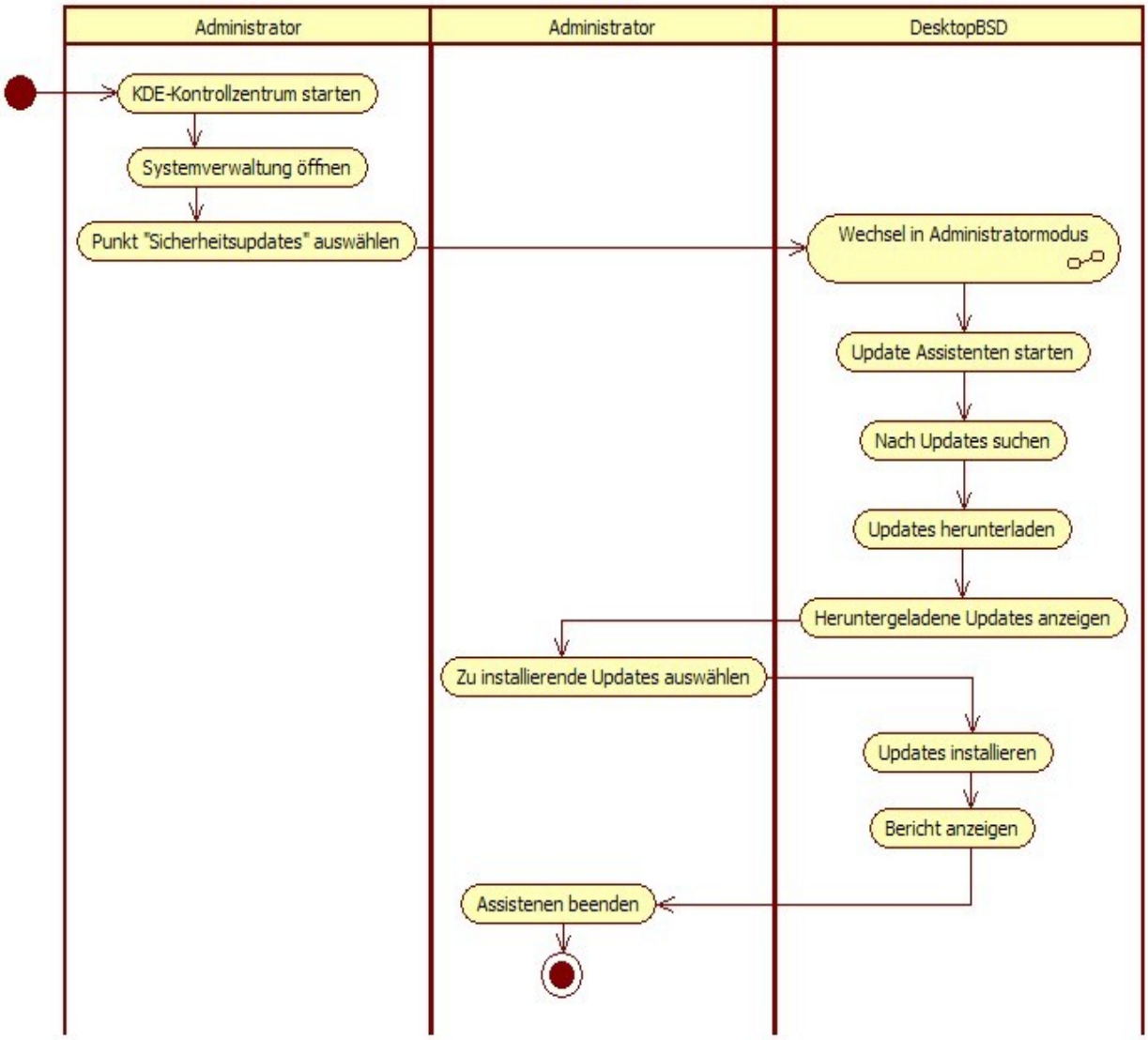
4.4.8 Anwendungsfall

UCD / Sicherheitsupdates



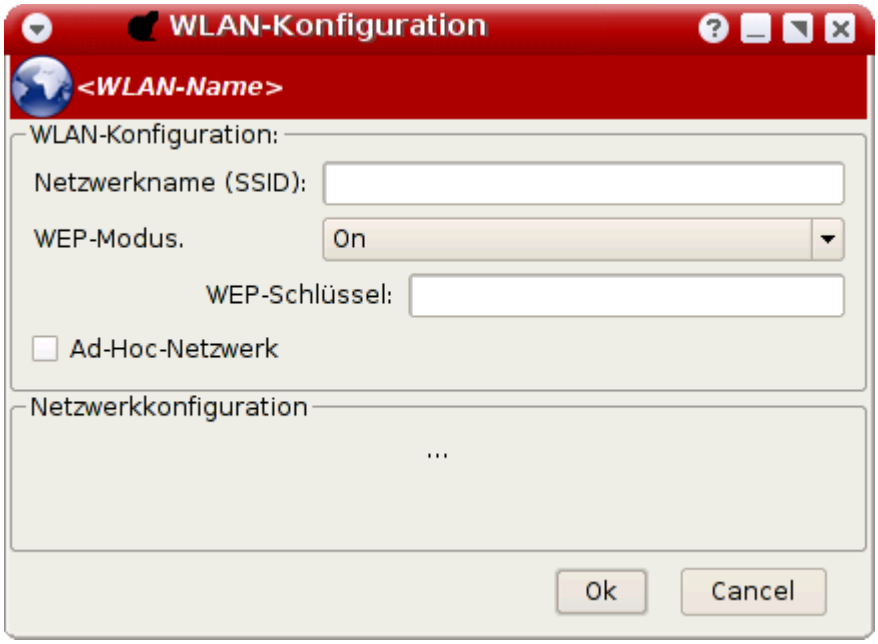
4.4.9 Aktivitätendiagramm

ACD / Sicherheitsupdates



4.5 Konfiguration von Ad-Hoc-WLAN

4.5.1 GUI-Prototypen



4.5.2 Funktion

(Siehe Klassendiagramm)

4.5.3 Umgebung

Der PowerConfigurator ist im KDE-Systemtray eingebettet und wird durch ein Symbol

(Trayicon) dargestellt. Der Benutzer kann nach Rechtsklick auf das Symbol in einem Menü bestimmte Parameter (zB minimale/maximale CPU-Taktung) einstellen oder mit Doppelklick die „vollwertige“ Verwaltung im Kontrollzentrum starten.

4.5.4 Schnittstellen

Funktion: Netzwerkinterfaces kontrollieren und konfigurieren

```
#include <sys/ioctl.h>

int
ioctl(int d, unsigned long request, ...);
```

d: Filedeskriptor eines Sockets zu dem Device

request: Auszuführende Operation

Dritter, optionaler Parameter: Datenstrukturen des Typs *ieee80211req*, *ifreq* und *ifmediareq*.

Auszug: Betriebssystem-Include net80211/ieee80211_ioctl.h

```
struct ieee80211req {
        char            i_name[IFNAMSIZ];       /* if_name, e.g. "wi0" */
        u_int16_t       i_type;                 /* req type */
        int16_t         i_val;                  /* Index or simple value */
        int16_t         i_len;                  /* Index or simple value */
        void            *i_data;                /* Extra data */
};
```

Auszug: Betriebssystem-Include net/if.h

```
/*
 * Interface request structure used for socket
 * ioctl's.  All interface ioctl's must have parameter
 * definitions which begin with ifr_name.  The
 * remainder may be interface specific.
 */
struct  ifreq {
        char    ifr_name[IFNAMSIZ];             /* if name, e.g. "en0" */
        union {
                struct  sockaddr ifru_addr;
                struct  sockaddr ifru_dstaddr;
                struct  sockaddr ifru_broadaddr;
                short   ifru_flags[2];
                short   ifru_index;
                int     ifru_metric;
                int     ifru_mtu;
                int     ifru_phys;
                int     ifru_media;
                caddr_t ifru_data;
                int     ifru_cap[2];
        } ifr_ifru;
#define ifr_addr        ifr_ifru.ifru_addr      /* address */
#define ifr_dstaddr     ifr_ifru.ifru_dstaddr   /* other end of p-to-p link */
#define ifr_broadaddr   ifr_ifru.ifru_broadaddr /* broadcast address */
```

```
#define ifr_flags       ifr_ifru.ifru_flags[0]  /* flags (low 16 bits) */
#define ifr_flagshigh   ifr_ifru.ifru_flags[1]  /* flags (high 16 bits) */
#define ifr_metric      ifr_ifru.ifru_metric    /* metric */
#define ifr_mtu         ifr_ifru.ifru_mtu       /* mtu */
#define ifr_phys        ifr_ifru.ifru_phys      /* physical wire */
#define ifr_media       ifr_ifru.ifru_media     /* physical media */
#define ifr_data        ifr_ifru.ifru_data      /* for use by interface */
#define ifr_reqcap      ifr_ifru.ifru_cap[0]    /* requested capabilities */
#define ifr_curcap      ifr_ifru.ifru_cap[1]    /* current capabilities */
#define ifr_index       ifr_ifru.ifru_index     /* interface index */
};
```

Auszug: Betriebssystem-Include net/if.h

```
struct ifmediareq {
        char    ifm_name[IFNAMSIZ];     /* if name, e.g. "en0" */
        int     ifm_current;            /* current media options */
        int     ifm_mask;               /* don't care mask */
        int     ifm_status;             /* media status */
        int     ifm_active;             /* active options */
        int     ifm_count;              /* # entries in ifm_ulist array */
        int     *ifm_ulist;             /* media words */
};
```

4.5.5 Realisierung

Als Programmiersprache wird C++ und das Qt-Framework verwendet.

Aus den KDE-Bibliotheken werden folgende Elemente verwendet:

- KSystemTray

4.5.5.1 Methoden

4.5.6 Lokale Daten

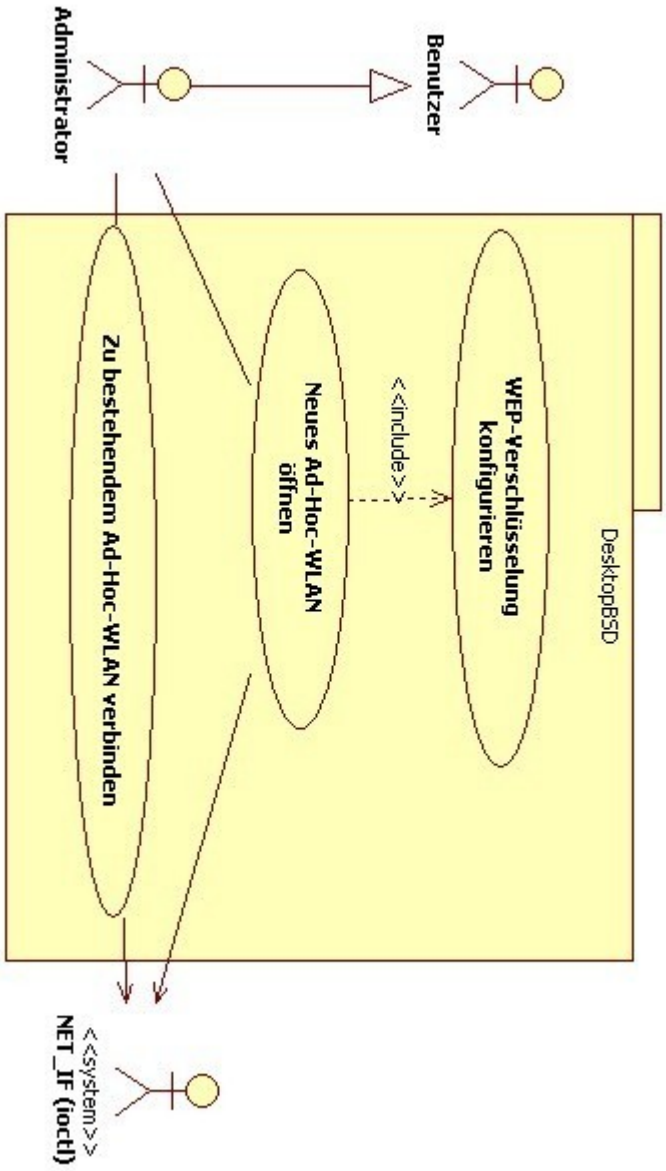
-

4.5.7 Ausnahmeverhalten

Wenn keine Verbindung zum WLAN-Adapter hergestellt werden kann, wird das Auswahlfeld der WLANs deaktiviert.

4.5.8 Anwendungsfall

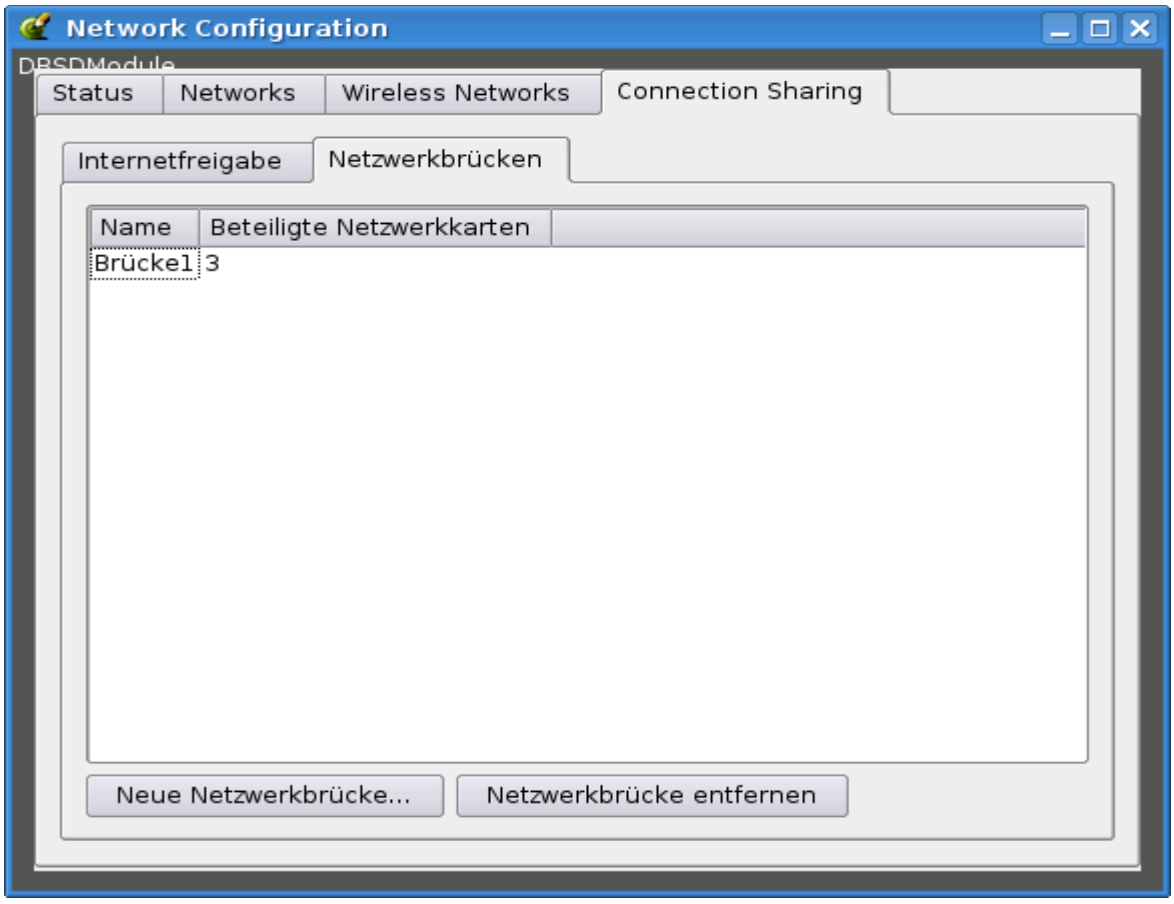
UCD / Ad-Hoc WLAN

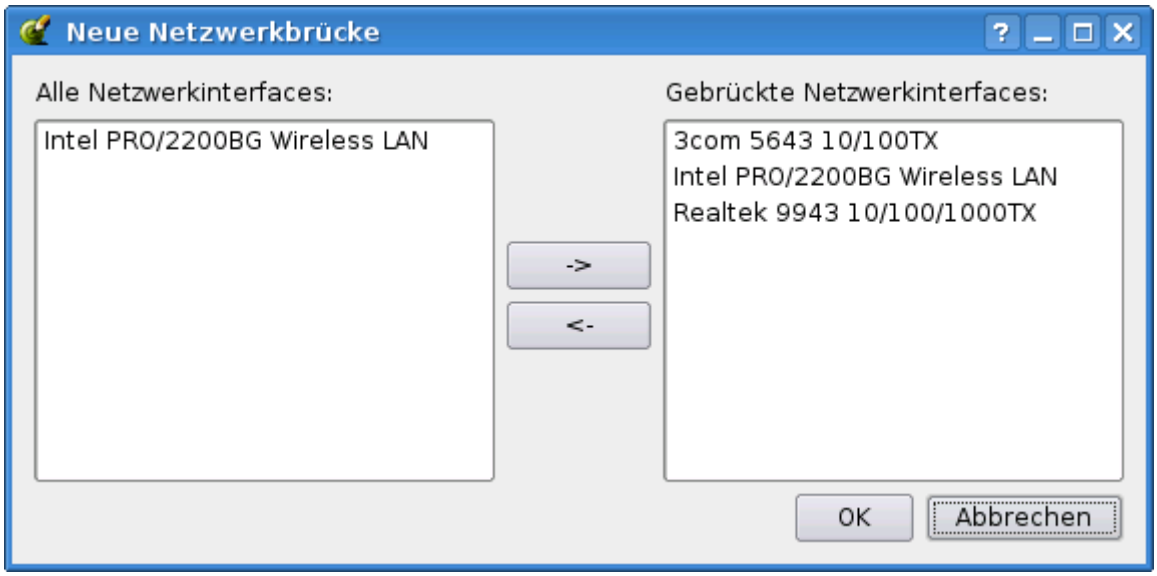


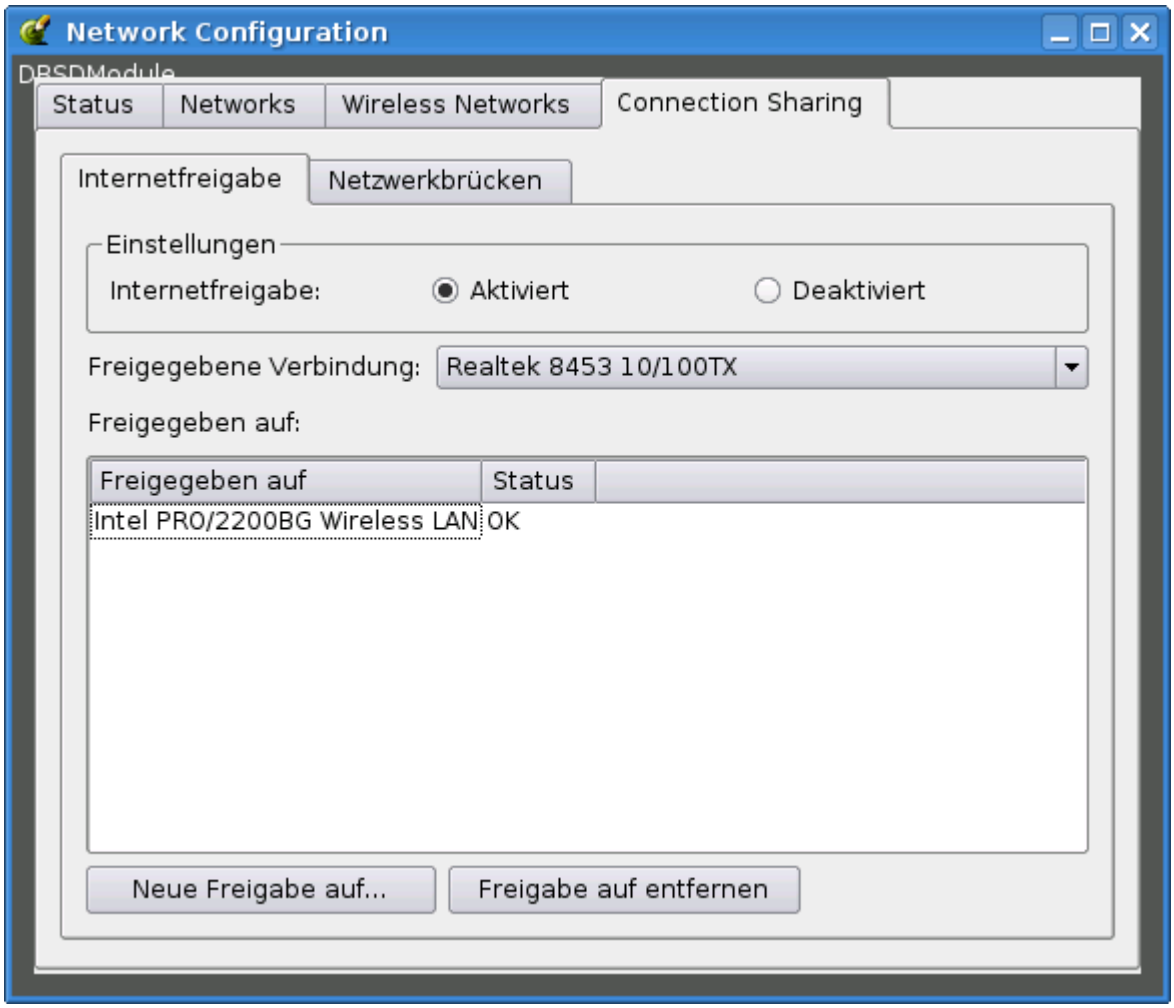
4.5.9 Aktivitätendiagramm

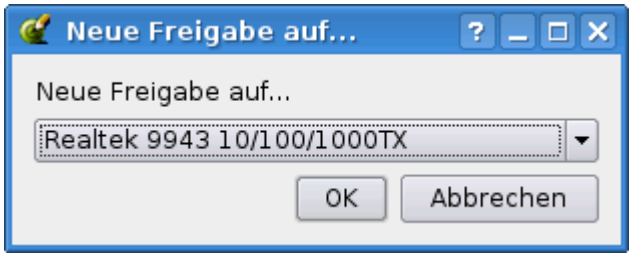
4.6 Internetverbindungsfreigabe

4.6.1 GUI-Prototypen









4.6.2 Funktion

(Siehe Klassendiagramm)

Die Klasse NetworkSharing bietet Möglichkeiten, über Bridging oder Routing zwei Netzwerkinterfaces zu „verbinden“, um von einem Netz auf das andere zugreifen zu können. NetInterface ist eine Netzwerkkarte, die entsprechende Methoden zum Setzen einer IP-Adresse etc. enthält.

Funktionsübersicht

- Netzwerkfreigabe mittels Routing erstellen
- Netzwerkfreigabe mittels Bridging erstellen
- Netzwerkfreigabe löschen
- Netzwerkfreigabestatus abrufen

4.6.3 Umgebung

Die Internetverbindungsfreigabe ist auch als Symbol in der Kontrollleiste eingebettet. Durch Doppelklick wird das „vollwertige“ Konfigurationsmodul aus dem Kontrollzentrum gestartet.

4.6.4 Schnittstellen

Funktion: Netzwerkinterfaces kontrollieren und konfigurieren

```
#include <sys/ioctl.h>

int
ioctl(int d, unsigned long request, ...);
```

d: Filedeskriptor eines Sockets zu dem Device

request: Auszuführende Operation

Dritter, optionaler Parameter: Datenstrukturen des Typs *ieee80211req*, *ifreq* und *ifmediareq*.

Auszug: Betriebssystem-Include net80211/ieee80211_ioctl.h

```
struct ieee80211req {
        char            i_name[IFNAMSIZ];       /* if_name, e.g. "wi0" */
        u_int16_t       i_type;                 /* req type */
        int16_t         i_val;                  /* Index or simple value */
        int16_t         i_len;                  /* Index or simple value */
        void            *i_data;                /* Extra data */
};
```

Auszug: Betriebssystem-Include net/if.h

```
/*
 * Interface request structure used for socket
 * ioctl's.  All interface ioctl's must have parameter
 * definitions which begin with ifr_name.  The
 * remainder may be interface specific.
 */
struct  ifreq {
        char    ifr_name[IFNAMSIZ];             /* if name, e.g. "en0" */
        union {
                struct  sockaddr ifru_addr;
                struct  sockaddr ifru_dstaddr;
                struct  sockaddr ifru_broadaddr;
                short   ifru_flags[2];
                short   ifru_index;
```

```
                int     ifru_metric;
                int     ifru_mtu;
                int     ifru_phys;
                int     ifru_media;
                caddr_t ifru_data;
                int     ifru_cap[2];
        } ifr_ifru;
#define ifr_addr        ifr_ifru.ifru_addr      /* address */
#define ifr_dstaddr     ifr_ifru.ifru_dstaddr   /* other end of p-to-p link */
#define ifr_broadaddr   ifr_ifru.ifru_broadaddr /* broadcast address */
#define ifr_flags       ifr_ifru.ifru_flags[0]  /* flags (low 16 bits) */
#define ifr_flagshigh   ifr_ifru.ifru_flags[1]  /* flags (high 16 bits) */
#define ifr_metric      ifr_ifru.ifru_metric    /* metric */
#define ifr_mtu         ifr_ifru.ifru_mtu       /* mtu */
#define ifr_phys        ifr_ifru.ifru_phys      /* physical wire */
#define ifr_media       ifr_ifru.ifru_media     /* physical media */
#define ifr_data        ifr_ifru.ifru_data      /* for use by interface */
#define ifr_reqcap      ifr_ifru.ifru_cap[0]    /* requested capabilities */
#define ifr_curcap      ifr_ifru.ifru_cap[1]    /* current capabilities */
#define ifr_index       ifr_ifru.ifru_index     /* interface index */
};
```

Auszug: Betriebssystem-Include net/if.h

```
struct ifmediareq {
        char    ifm_name[IFNAMSIZ];     /* if name, e.g. "en0" */
        int     ifm_current;            /* current media options */
        int     ifm_mask;               /* don't care mask */
        int     ifm_status;             /* media status */
        int     ifm_active;             /* active options */
        int     ifm_count;              /* # entries in ifm_ulist array */
        int     *ifm_ulist;             /* media words */
};
```

4.6.5 Realisierung

Als Programmiersprache wird C++ und das Qt-Framework verwendet.

Aus den KDE-Bibliotheken werden folgende Elemente verwendet:

- KSystemTray

4.6.5.1 Methoden

Bridge

- *QPtrList getNICs()*

  Gibt eine Liste aller überbrückten Netzwerkgeräte einer Bridge zurück
- void addNIC(NetInterface* nic)

  Fügt der Bridge ein weiteres Netzwerkgerät hinzu.
- void setNIC(QPtrList<NetInterface> nics)

  Setzt die Liste aller Netzwerkgeräte.

- void setActivated(bool activated)

  Aktiviert bzw. deaktivierte eine Netzwerkbrücke.

- bool getActivated()

  Gibt „TRUE“ zurück wenn die Netzwerkbrücke aktiviert ist und „FALSE“ wenn sie deaktiviert ist.

BridgedNetConnections

- *QPtrList<NetInterface> getBridges()*

  Gibt alle vorhandenen Netzwerkbrücken zurück.

- *void setBridges(QPtrList<NetInterface> bridge)*

  Setzt die Liste aller Netzwerkbrücken.

- void addBridge(const Bridge & bridge)

  Fügt eine neue Netzwerkbrücke in die Liste aller Netzwerkbrücken ein.

- void removeBridge(const Bridge & bridge)

  Entfernt eine Netzwerkbrücke.

- BridgedNetConnection getInstance()

  Diese Methode ist Teil des Singleton-Konzeptes und ist dafür verantwortlich, dass die Klasse nur eine Instanz haben kann. Sie gibt die Instanz der Klasse „BridgedNetConnection“ zurück.

RoutedNetConnection

- *QPtrList getAllNICs()*

  Gibt eine Liste aller Netzwerkgeräte zurück die über eine „Routing“ Verbindung verbunden sind.

- *void setSourceNIC(NetInterface *_sourceNIC)*

  Speichert Informationen über die Netzwerkkarte, die die Verbindung freigibt.

- NetInterface getSourceNIC()

  Gibt Informationen über die Netzwerkkarte, die die Freigabe benutzt, zurück.

- void addDestinationNIC(NetInterface *_destinationNIC)

  Speichert Informationen über die Netzwerkkarte, die die Freigabe benutzt.

- void removeDestinationNIC(NetInterface *_destinationNIC)

  Entfernt ein Netzwerkgerät, dass eine Freigabe bentutzt.

- QPtrList<NetInterface> getDestinationNICs()

  Gibt eine Liste der Netzwerkgeräte zurück, die die Freigabe benutzen.

- void setActivated(bool activated)

Aktiviert bzw. deaktiviert eine „Routing“ Verbindung zw. Netzwerkgeräten.

- RoutedNetConnection getInstance()

  Diese Methode ist Teil des Singleton-Konzeptes und ist dafr verantwortlich dass die Klasse nur eine Instanz haben kann. Sie gibt die Instanz der Klasse „BridgedNetConnection“ zurück.

4.6.6 Lokale Daten

RoutedNetConnection

- private:
  - NetInterface sourceNIC;
  - QPtrList<NetInterface> destinationNIC;
  - static RoutedNetConnection instance;

SharedNetConnections

- private:
  - QPtrList<NetInterface> bridges;
  - static RoutedNetConnection instance;

Bridge

- private:
  - QPtrList<NetInterface> nics;
  - bool activated;

Device

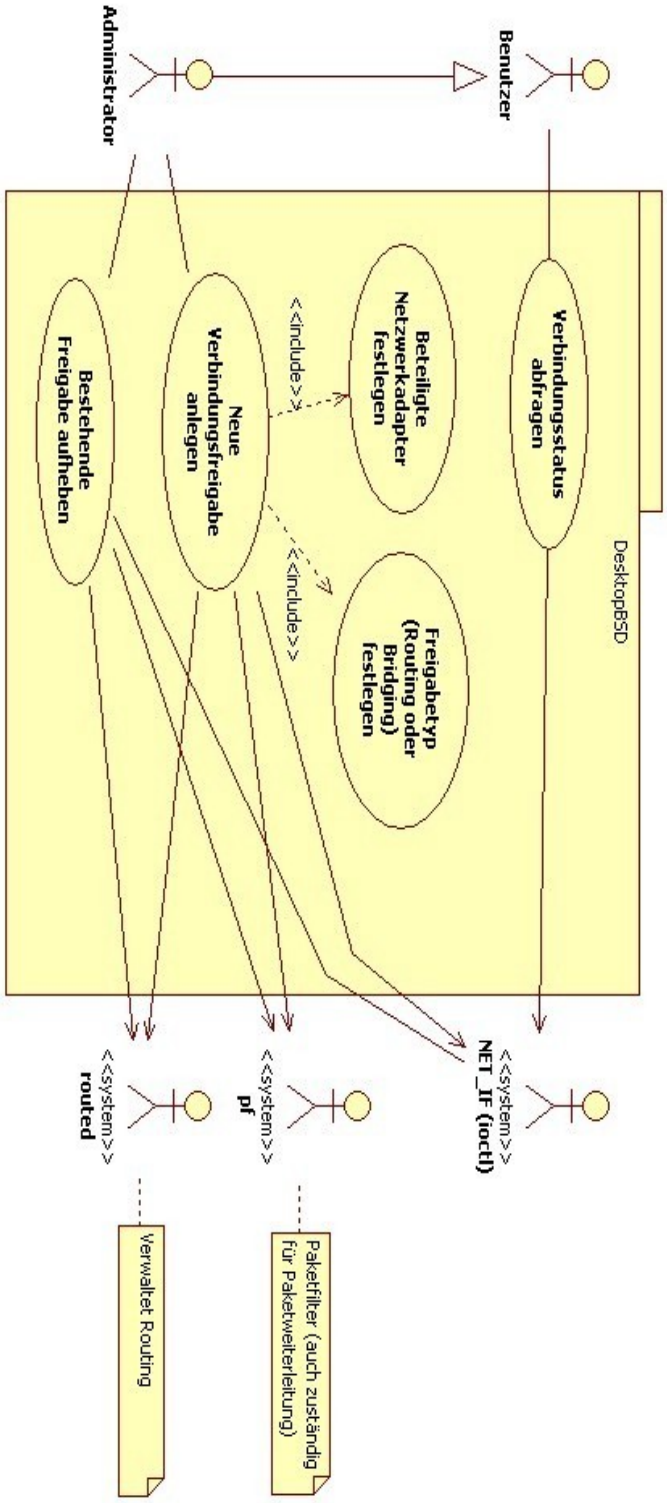
- private:
  - QString name;
  - QString desc;
  - QString driver;

4.6.7 Ausnahmeverhalten

-

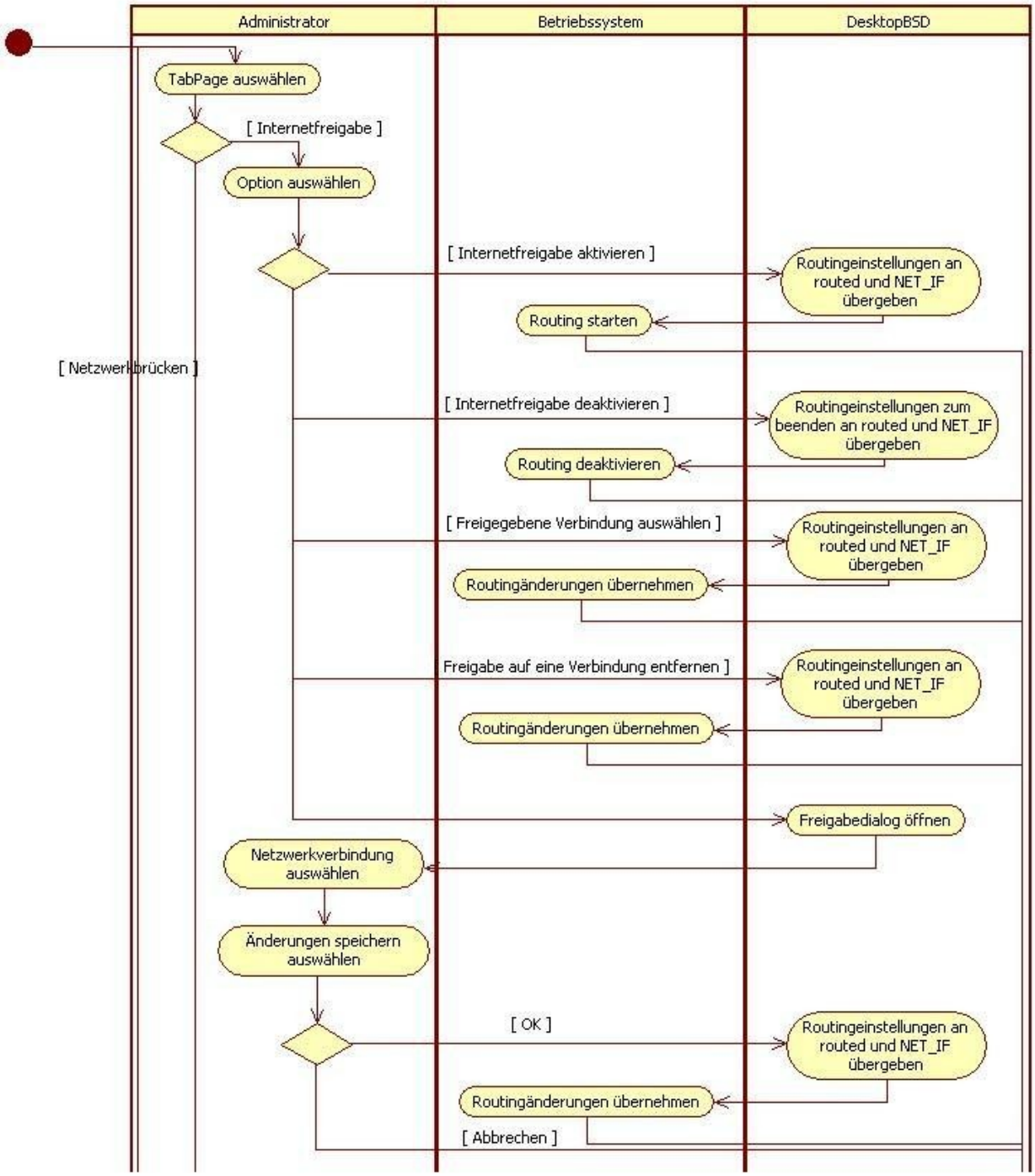
4.6.8 Anwendungsfall

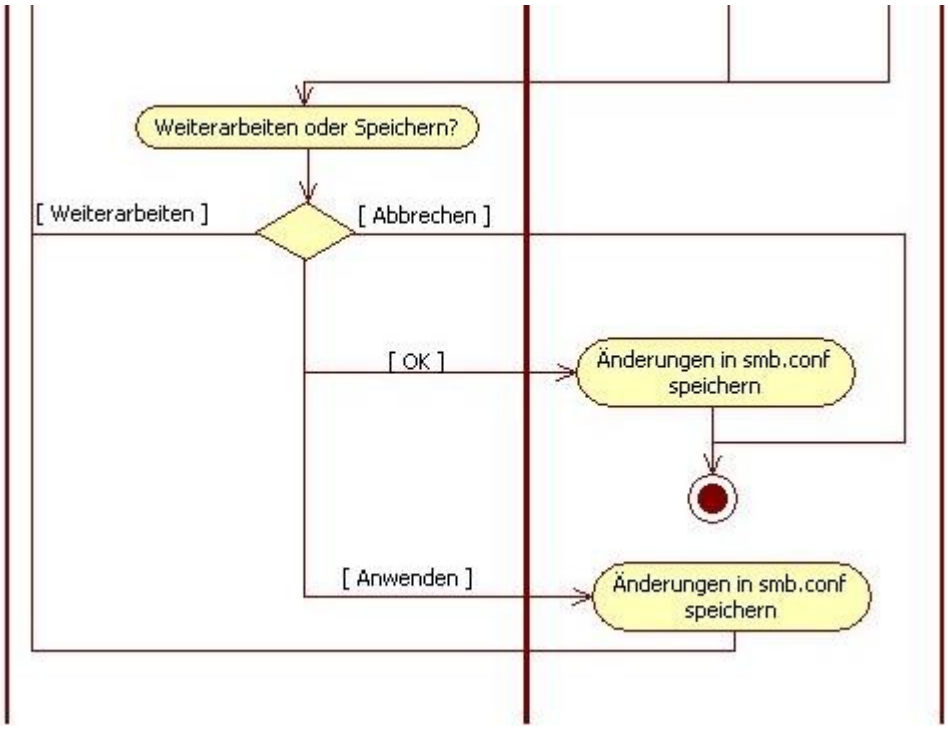
UCD / Verbindungsfreigabe



4.6.9 Aktivitätendiagramm

ACD / Verbindungsfreigabe




Weiterarbeiten oder Speichern?
[Weiterarbeiten]
[Abbrechen]
[OK]
Änderungen in smb.conf speichern
[Anwenden]
Änderungen in smb.conf speichern

4.7 Zeitabgleich mit Timeserver

4.7.1 GUI-Prototypen

4.7.2 Funktion

Mithilfe dieses Tools kann die Systemuhrzeit mit einem Timeserver aus dem Internet synchronisiert werden. Der Timeserver kann dabei aus einer Liste, die aus dem Internet abgerufen wird, ausgewählt oder manuell eingegeben werden.

(Siehe Klassendiagramm)

NTPClientEngine ist eine Klasse, die den Befehl *ntpdate* verwendet, um von einem entfernten NTP-Zeitserver die aktuelle Zeit abzurufen und als Systemzeit zu setzen. *ModTimeUpdate* erlaubt das Einstellen des Zeitservers.

4.7.3 Umgebung

Die Zeitsynchronisation ist im „Kontrollzentrum“ des KDE eingebettet und wird aus diesem heraus aufgerufen.

4.7.4 Schnittstellen

Als einzige Schnittstelle wird hier das Systemprogramm „ntpdate“ verwendet. Dieses funktioniert sehr einfach. Es wird nur die IP-Adresse oder der Name des Servers als Parameter übergeben. „ntpdate“ synchronisiert dann die Uhrzeit bzw. das Datum mit dem des Servers. Das ganze geschieht über das NTP-Protokoll.

Beispiel:

```
# ntpdate 80.64.135.105
20 Apr 22:39:16 ntpdate[90655]: adjust time server 80.64.135.105
offset 0.254673 sec
```

4.7.5 Realisierung

Als Programmiersprache wird C++ und das Qt-Framework verwendet.

4.7.5.1 Methoden

NTPClientEngine:

- *int updateSystemTime(const QUrl & server)*

  Synchronisiert die Systemuhrzeit mit dem Timeserver, dessen URL als Parameter mitgegeben wird.

4.7.6 Lokale Daten

private:

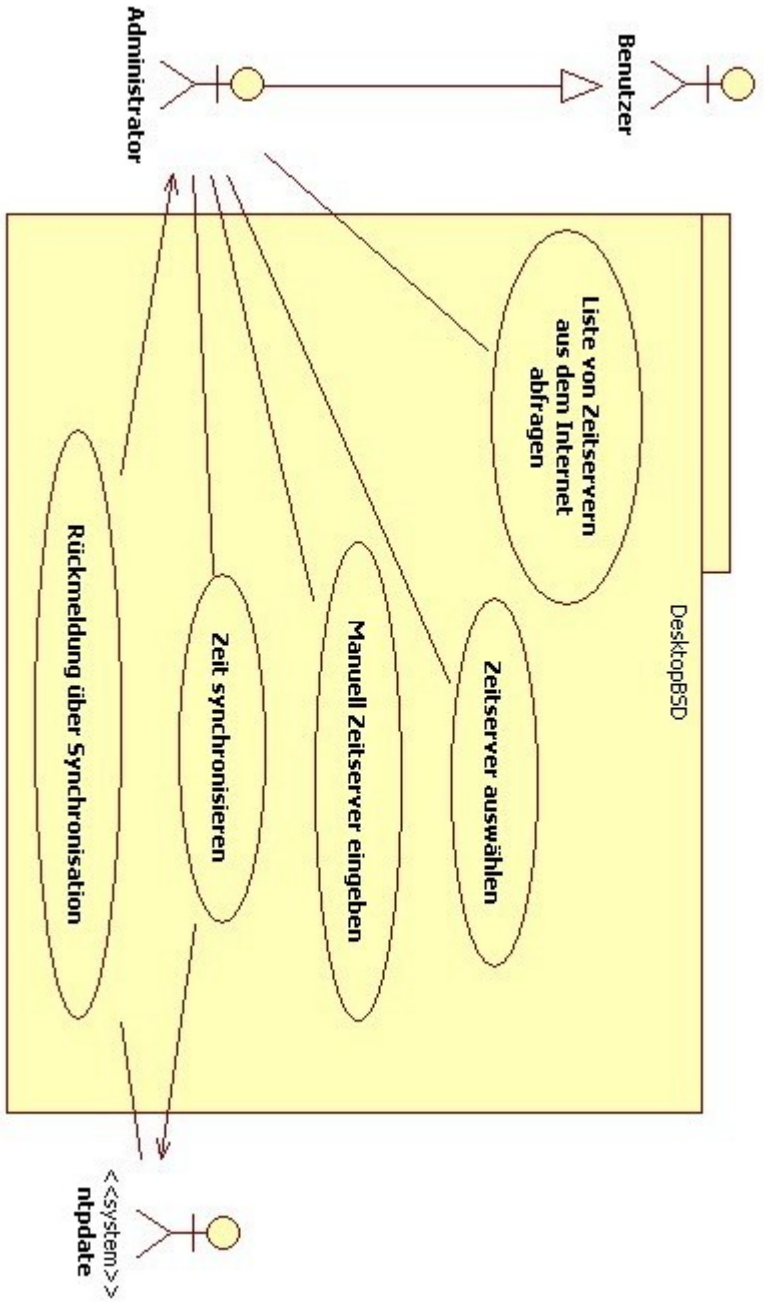
QURL server; // URL des Zeitservers

4.7.7 Ausnahmeverhalten

Falls der angegebene Server nicht erreichbar ist, wird eine Benachrichtigung angezeigt. Die angezeigte Nachricht beinhaltet die Prozessausgabe von „ntpdate“, welche mittels der Qt-Klasse „QProcess“ ausgelesen werden kann.

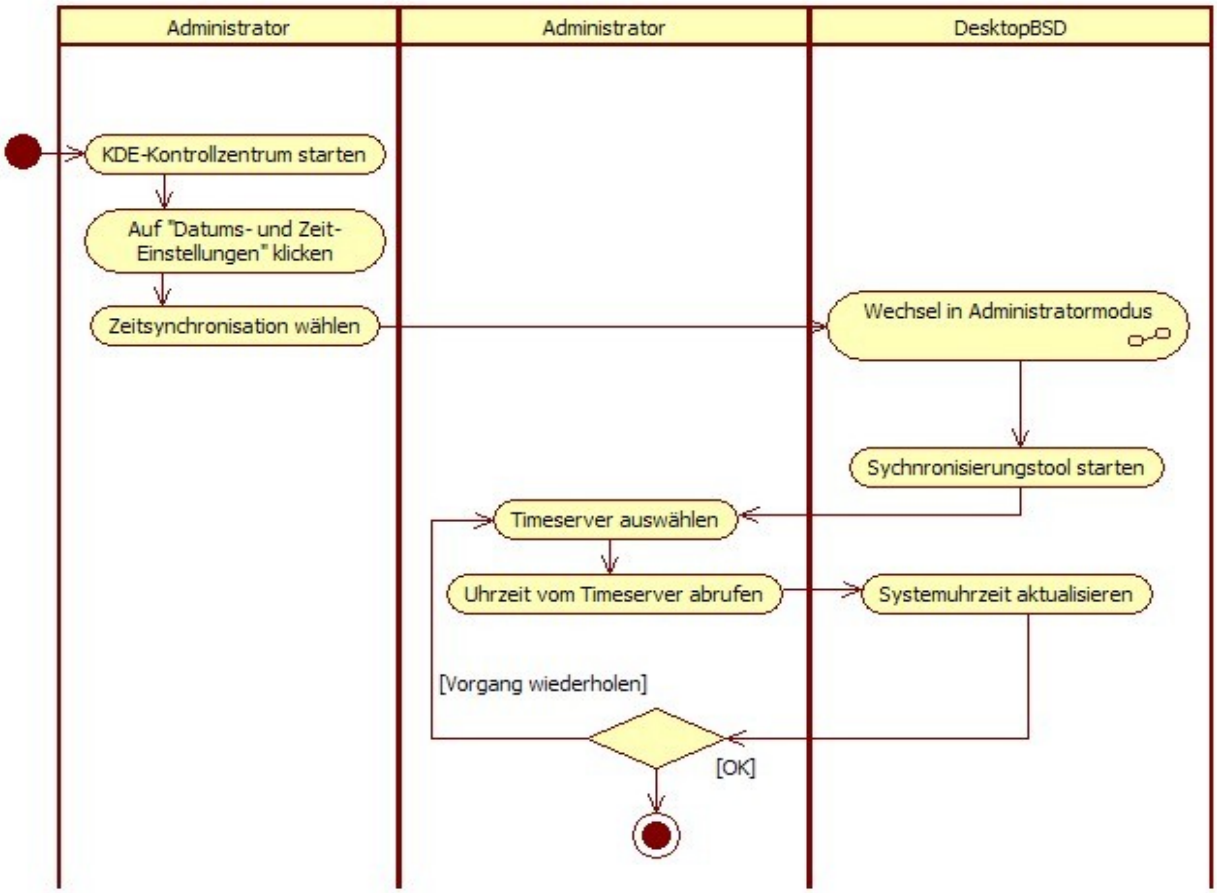
4.7.8 Anwendungsfall

UCD / Zeitabgleich



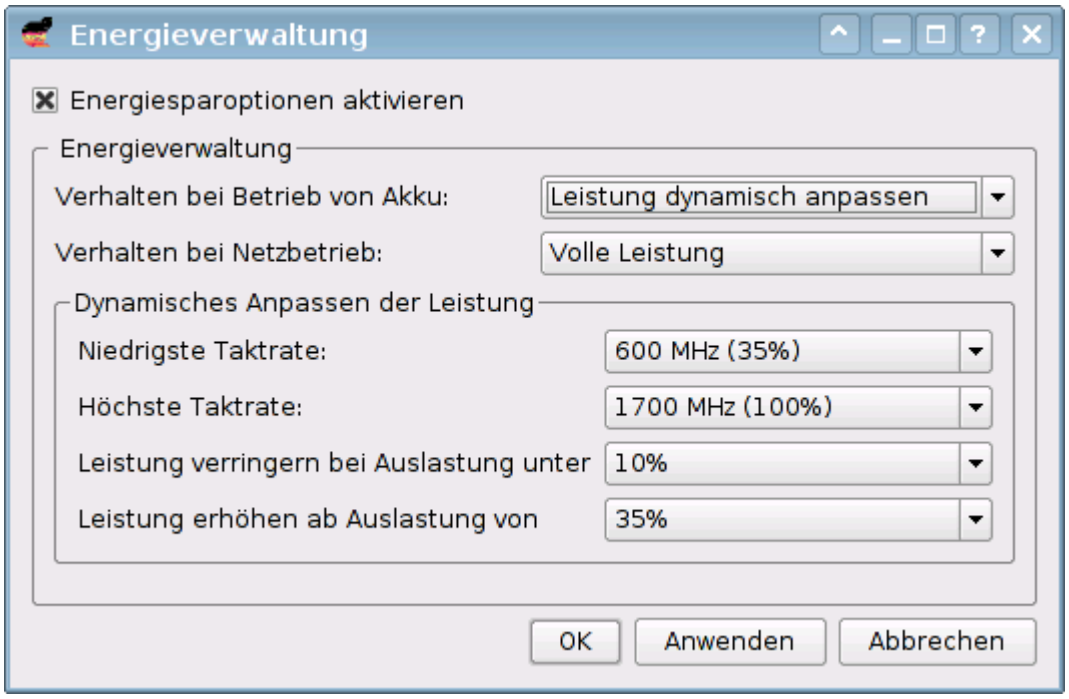
4.7.9 Aktivitätendiagramm

ACD / Zeitabgleich



4.8 Konfiguration von Energieoptionen

4.8.1 GUI-Prototypen



4.8.2 Funktion

Dieses Tool ermöglicht es, Energiesparoptionen zur Laufzeit zu aktivieren und die Systemleistung dynamisch an den Akku- oder Netzbetrieb anzupassen (z.B.: Taktrate, Auslastung, ...)

(Siehe Klassendiagramm)

- PowerConfigurator setzt die entsprechenden Einstellungen für powerd, um die Regelung der Leistung vorzunehmen.
- ModPowerControl ist eine grafische Oberfläche, mit der die Parameter der Leistungsanpassung festgelegt werden können (siehe GUI-Prototyp).
- BatteryTray ist ein Symbol im Systembereich der Kontrollleiste (Tray-Icon), das den Typ der Stromversorgung (Akku oder Netzbetrieb) und den Akkustand anzeigt. Weiters kann mit Doppelklick auf dieses Symbol oder das Rechtsklickmenü die Energieverwaltung (ModPowerControl) aufgerufen werden.

4.8.3 Umgebung

Der PowerConfigurator ist im KDE-Systemtray eingebettet und wird durch ein Symbol (Trayicon) dargestellt.

4.8.4 Schnittstellen

- Änderung der Konfigurationsdatei *rc.conf*
  - Variable *powerd_enable*
    - *powerd_enable="YES"* aktiviert Energiesparmechanismen
    - *powerd_enable="NO"* (Standard) deaktiviert Energiesparmech.
  - Variable *powerd_flags*
    - *-a, -b, -n:* Verwendetes Profil bei Netzteilbetrieb, Akkubetrieb, nicht erkennbarem Zustand
      - *max*: Maximale Leistung
      - *min*: Kleinstmögliche Leistung
      - *adaptive*: Dynamisches Anpassen je nach Leistungsbedarf
    - *-i, -r:* Auslastung, bei der dynamisch hinunter-/hochgetaktet werden soll
    - *-p:* Intervall in ms, in dem der Wert abgefragt werden soll

Beispiel:

```
powerd_enable="YES" # Energiesparmechanismus aktiviert

# Bei Netzteilbetrieb maximale Leistung, bei Akkubetrieb dynamisch
# angepasst, bei 33% Leerlauf hochtakten, bei 66% Leerlauf hinuntertakten,
# Wert alle Sekunden abfragen
powerd_flags="-a max -b adaptive -r 33 -i 66 -p 1000"
```

4.8.5 Realisierung

Als Programmiersprache wird C++ und das Qt-Framework verwendet.

Aus den KDE-Bibliotheken werden folgende Elemente verwendet:

- KSystemTray

4.8.5.1 Methoden

PowerConfigurator:

- *int setEnable(bool enabled)*

  Aktiviert die Energieoptionen wenn Parameter „enabled" gleich „True" ist bzw. deaktiviert diese bei „False".

- *int setModePower(Mode mode, PowerMode power)*

  Für den Betrieb von einer bestimmten Stromquelle die verwendete Konfiguration (minimale, maximale oder dynamisch angepasste Leistung) setzen.

- *int setUpdateInterval(uint ms)*

  Setzt das Aktualisierungsintervall, in dem powerd prüft, ob leistungsabhängig hoch oder hinunter getaktet werden muss.

- int setDecreaseLoad(uint load)

  Setzt die Systemauslastung, die erforderlich ist, um die CPU hochzutakten.

- int setIncreaseLoad(uint load)

  Setzt die Systemauslastung, die erreicht werden muss, um die CPU hinunterzutakten.

4.8.6 Lokale Daten

PowerConfigurator:

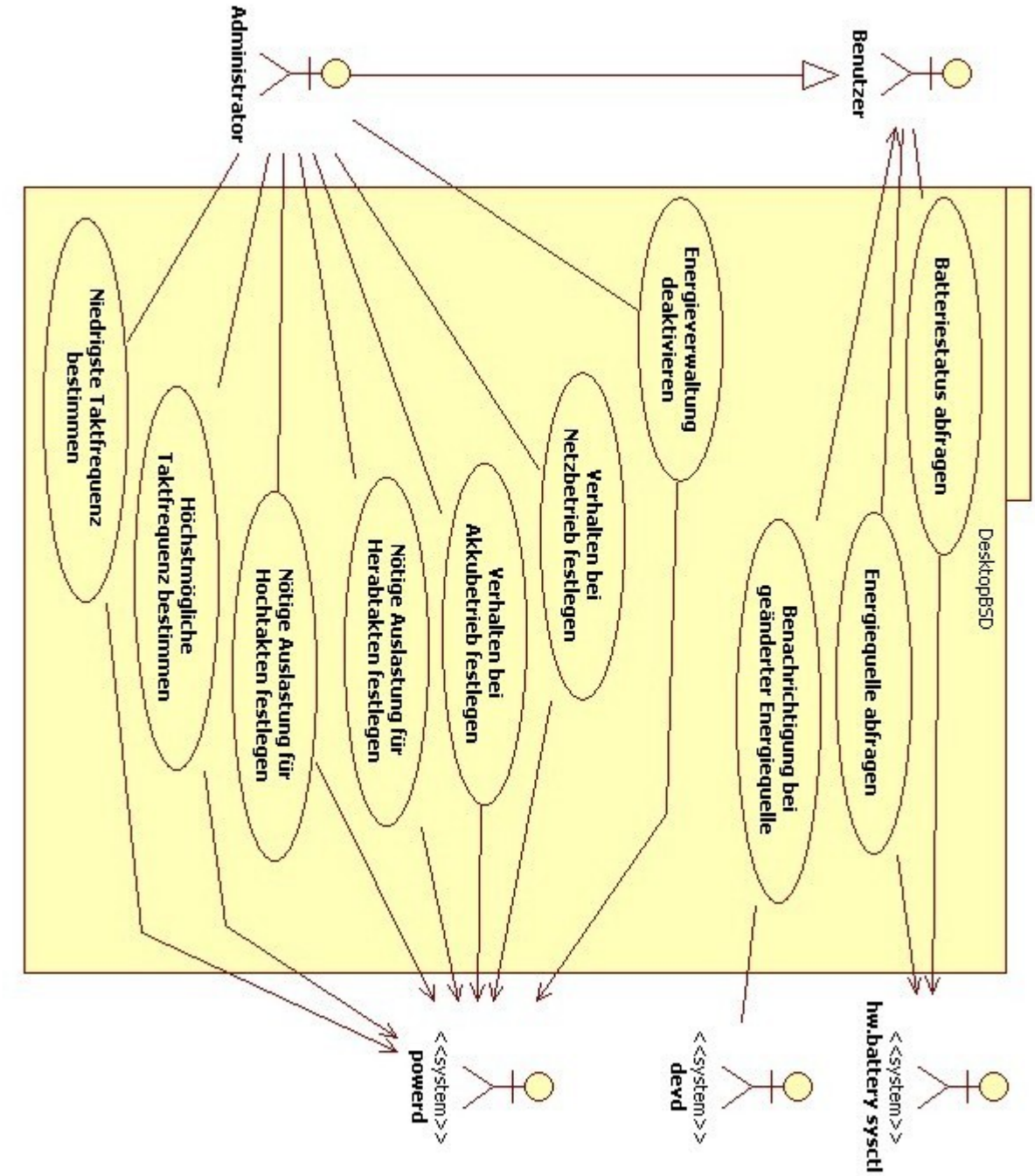
```
private:
    enum Mode {
        ACLine,
        Battery,
        Unknown
    };

    enum PowerMode{
        Minimum,
        Adaptive,
        Maximum
    };

    bool enabled;       // Ob die Energiesparoptionen aktiviert sind
    Mode mode;          // Die aktuelle Stromversorgung: ACLine, Battery
                                                        oder  Unknown
    PowerMode power;    // Entweder Minimum, Adaptive oder Maximum
    uint ms;            // Zeitintervall (Millisekunden) zur Aktualisierung
                                                        der Anzeige
    uint load;      // CPU-Auslastungswert ab der runtergetaktet
                    bzw. hochgetaktet wird
```

4.8.7 Ausnahmeverhalten

Kommt es durch bestimmte Aktionen zu einer Ausnahme, wechselt der powerd automatisch in den Status „unknown“. In diesem Fall wird dem Benutzer eine Benachrichtigung (mithilfe eines KPassivePopup) angezeigt.

4.8.8 Anwendungsfall

UCD / Energieoptionen



4.8.9 Aktivitätendiagramm

ACD / Energieoptionen