Pool Video Switch

Benutzer- und Entwicklerdokumentation

Entwickler-Team

Projekt des Lehrstuhls für Kommunikationssysteme

Rechenzentrum / Institut für Informatik

April 2010

back of titlepage

Pool Video Switch

Benutzer- und Entwicklerdokumentation

OpenSLX Project
Freiburg
2010

.
OpenSLX Project
Lehrstuhl für Kommunikationssysteme
Rechenzentrum & Institut für Informatik an der Technischen Fakultät
Herrmann-Herder-Str. 10
79104 FREIBURG i.Br.
pvs@uni-freiburg.de

Inhaltsverzeichnis

Zusammenfassung

Der Pool Video Switch (PVS) ist eine Applikation, die seit einiger Zeit im Zuge des OpenSLX-Projekts am Rechenzentrum der Universität Freiburg und am Lehrstuhl für Kommunikationssysteme entwickelt wird.
Inzwischen liegt die Version 3 vor ...

Teil I

Idee und Überblick

1 Idee

Video-Switches für Klassenräume sind schon seit langem auf dem Markt: Sie setzen typischerweise auf eine Hardware-Lösung, die das Verlegen von vielen Metern Kabel bei dadurch erzwungener statischer Anordnung der Rechner verlangt. Sie sind nicht nur aufgrund des Einsatzes spezieller Hardware relativ teuer, sondern unflexibel bei technischen Weiterentwicklungen: Höhere Bildschirmauflösungen oder der Wechsel vom VGA auf den DVI oder PS2 auf den USB-Stecker sind nur durch einen Komplettaustausch oder unangenehme Kompromisse zu haben. Ziemlich ungeeignet erweisen sich hardware-basierte Lösungen beim Einsatz mobiler Geräte wie Laptops. Zudem erschweren sie die Veränderung der ursprünglichen Aufstellung der Maschinen und schaffen damit ungünstige Bedingungen für neue didaktische Konzepte. Neben den Hardware-basierten Varianten existieren inzwischen eine Reihe von

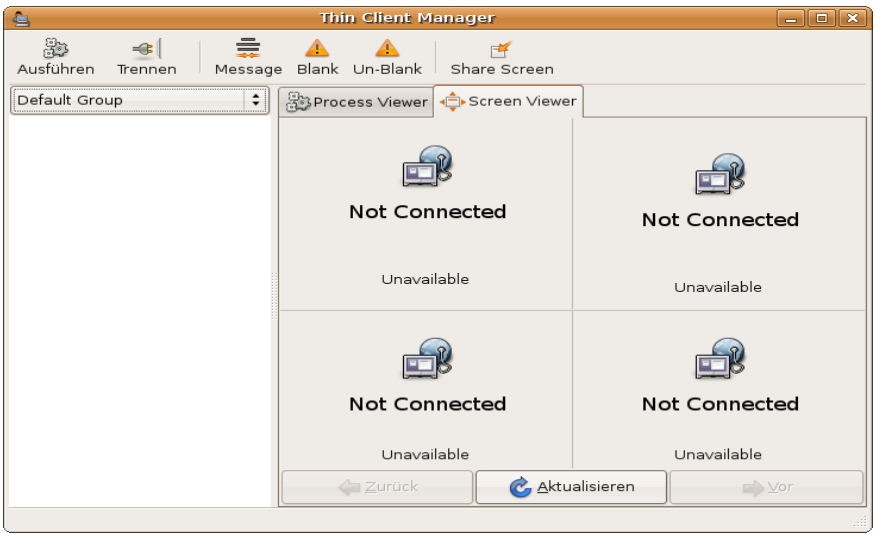

Abbildung 1.1: Student Control Panel des Edubuntu-Pakets, welches im Umfeld von Linux-Terminalservern eingesetzt werden kann.

Software-Produkten. Leider bieten sie keine optimalen Lösungen: Neben oft exorbitanten Lizenzkosten, die oft mit einem didaktisch und konzeptionell fragwürdigen Feature-Set begründet werden, orientieren sie sich an einem einzigen auf den Schulungs-PCs festinstalliertem Betriebssystem. Oft stehen sie nur für ausgewählte Microsoft-Betriebssysteme zur Verfügung und können damit nicht dem Anspruch einer breiten und allgemeinen universitären Umgebung entsprechen. Zudem kommen regelmäßige, typischerweise kostenpflichtige Produkt-Updates

hinzu, denen sich nur begrenzt ausweichen läßt.
Dem soll ein eigener Ansatz entgegengesetzt werden, der die eingangs genannten Beschränkungen aufhebt und dem Anspruch einer modernen universitären computer-gestützten Lehre entgegenkommt. Das Projekt soll stufenweise von einer einfachen Basislösung in Rückmeldung mit den Lehrenden weiterentwickelt werden. Ein Open-Source-Ansatz gewährleistet zudem eine größere Nachhaltigkeit. Selbst bei einer Projekteinstellung können andere auf den Code weiterhin zurückgreifen und damit einerseits bestehende Installationen aktualisieren und andererseits eigene Entwicklungen schneller vorantreiben.
Bei der Entwicklung steht nicht die Neuerfindung bereits vorhandener Technologien, sondern deren intelligente Verknüpfung im Vordergrund. So bieten bereits Techniken wie VNC oder Xorg/X11 viele notwendige Grundlagen, die um pfiffige Werkzeuge mit intuitiven Oberflächen erweitert werden sollen. Eine gute Anschauung bietet beispielsweise die Open Source Software "Student Control Panel" des Edubuntu-Linux (2.1).

2 Funktionen eines Software-Video-Switchs

Damit ein Software-Video-Switch Hardwarelösungen ersetzen kann, muss zumindest eine gewisse Grundmenge an Funktionalität bereitgestellt werden. Diese Funktionen lassen sich in weiteren Schritten erweitern und können dabei über die der Hardwareimplementierungen hinausgehen. Gegebenenfalls muss eine Softwarelösung sich um Aspekte kümmern, die bei einer Hardwarelösung kaum ein Problem darstellen. Hierzu zählt das Thema der Abhörsicherheit.

2.1 Wünschenswerte Basisfunktionen

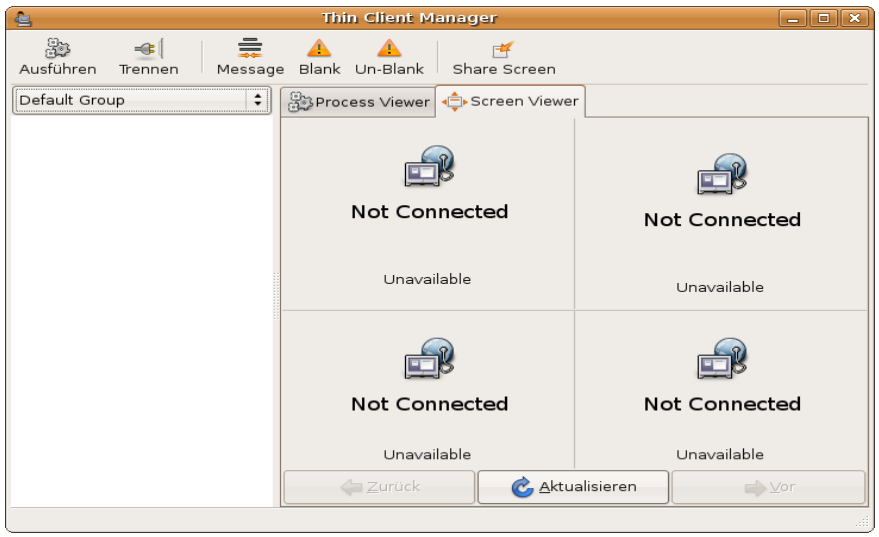

Abbildung 2.1: Student Control Panel des Edubuntu-Pakets, welches im Umfeld von Linux-Terminalservern eingesetzt werden kann.

2.2 Funktionserweiterungen

3 Aufbau dieses Handbuchs

Das Handbuch ist in drei Teile gegliedert, wobei der erste Teil mit diesen Erläuterungen bereits abgeschlossen wird. Der zweite Teil wendet sich an die Endanwender dieses Projekts, der dritte Teil umfasst die Entwicklerdokumentation.

3.1 Heraushebungen und Formatierungen

Zur besseren Lesbarkeit gibt es einige Konventionen zur Formatierung von einfachen Kommandos, Kommandozeilen oder Verzeichnis- und Dateinamen. Ebenso sind Skripte oder Auszüge aus C++-Quelltexten zur Heraushebung anders gesetzt.
Hierbei gelten folgende Regelungen: Ein einfaches Kommando wird **fett** gesetzt, wie beispielsweise **pvsmgr**. Dieses lässt sich in einem Terminalfenster absetzen oder über die Ausführenoption der grafischen Desktops. Kommandos mit Optionen und bei Bedarf Parametern werden so wie Skripten und Quelltexte in `Schreibmaschinenschrift` gesetzt aber nicht aus dem allgemeinen Textsatz herausgehoben. Solche Heraushebungen werden jedoch ab zwei Druckzeilen aufwärts sinnvoll. Verzeichnis- und Dateinamen, wobei es bei letzteren dann nicht um den Programmnamen, sondern Quelltexte, Konfigurationen etc. geht, werden *kursiv* gedruckt, wie beispielsweise */etc/pvs*.

Teil II

Bedienungsanleitung

4 PVS-Steuerkonsole

Die PVS-Steuerkonsole auch PVS-Manager genannt, wurde so benutzerfreundlich aufgebaut, dass die wichtigsten Funktionen durch einen Mouseklick entfernt werden. wir wollen in diesem Teil die unterschiedlichen Funktionen, die von der PVS-Steuerkonsole aus angeboten werden, vorstellen.

4.1 Allgemein

Die Abbildung 4.1 zeigt den PVS-Manager in seinem Gesamtbild. Ganz links steht die Liste der verbundenen Clients und rechts befinden sich die entsprechenden VNC-Frames.

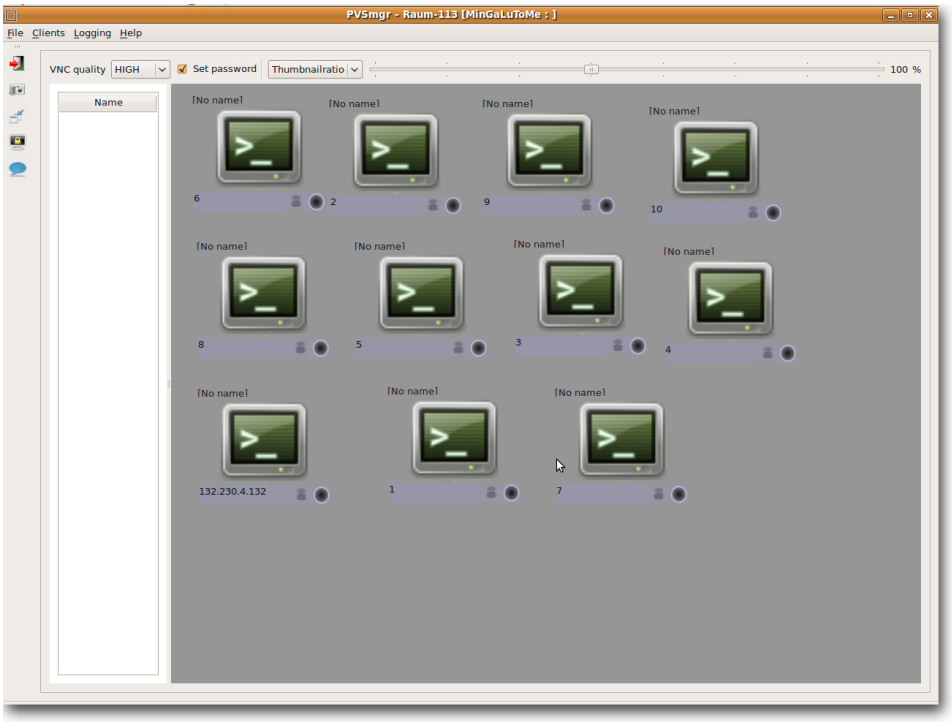

Abbildung 4.1: Die PVS-Steuerkonsole

4.1.1 Die Client-Liste

Die angezeigten Clients in dieser Liste haben mit dem Server eine erfolgreiche Verbindung aufbauen können. Mit dem entsprechenden Tastenkürzel wie es im 4.1.3 aufgelistet wird, kann man die Clients in unterschiedlichen Bezeichnung anzeigen lassen. Beim Rechtsklick auf die einzelnen Clients sind folgenden Aktionen möglich:
Lock Client(s): Damit werden die selektierten Clients gesperrt und dabei werden der Superclient und ein möglicher auf den Superclient projektierten Client ausgeschloßen.
Unlock Client(s): Die gesperrten Clients werden wieder entsperrt.
MsgLock Client(s): Idem wie beim *Lock Client(s)* aber hier bekommt der Client vor der Sperrung eine Nachricht vom pvsmgr.
MsgLock All Client(s): Ein *MsgLock Client(s)* für alle Clients.
Message Clients: Nachricht an den selektierten Clients schicken.
Projection: Es wird benutzt, um den VNC-Frame eines Clients zu einem anderen zu verschicken. Die Ausführung der Projektion verlangt, dass nur einen einzigen Client selektiert wird. Der selektierte Client ist die Quelle der Projektion. Die Ziele der Projektion werden über ein Popup-Fenster ausgewählt. Es wird auch möglich einen Client in einem schon existierenden Projektion mit einbeziehen zu lassen. Es ist natürlich nicht möglich ein Client gleichzeitig als Quelle und Ziel einer Projetion zu haben.
Unprojektion: Wird ausgewählt um eine existierende Projektion zu beenden. Ausgeführt auf eine Quelle (source-projection) einer Projektion werden alle betroffenen Clients (target-projection) wieder freigegeben. Die Quelle wird dann im Anschluß auch freigegeben.

4.1.2 Die VNC-Frames

Die VNC-Frames werden im ConnectionWindow angezeigt. Für den jeweiligen verbundenen Client wird ein kleines Fenster mit änderbarer Größe bereitgestellt. Die verfügbaren Aktionen per Rechtsklick auf die vorhandenen VNC-Frames sind ähnlich wie im 4.1.1 beschrieben wurde. Im ConnectionWindow kann man zwei Arten von Frames unterscheiden. Die Abbildung 4.2 zeigt einen *Dummy-* und einen *Nicht Dummy-Frame*.

Dummy-Frame: Ein *Dummy-Frame* lässt sich mit einem schwarzen Punkt rechts unten unterscheiden. Ein *Dummy-Frame* ist eine Art Template oder besser ein Platzhalter für den eigentlichen VNC-Frame. Man kann sie erzeugen oder löschen. Die Erzeugung erfolgt durch einen Rechtsmouseklick auf dem ConnectionWindow und das Löschen durch den Rechtsklick auf dem betroffenen Client. Die *dummy-Clients* werden meistens zur Definition von Profilen erzeugt. Sie werden also erzeugt und je nach Wunsch angeordnet und unter dem Menü *Profile manager* wird ein Dialog gestartet, in dem man das Profil nach dem Klick auf dem Knopf *new* unter den gewunschten Name speichern kann. Ausserdem unter dem Menü *Load profil* kann man ein existierendes Profil laden.

Nicht dummy-Frame: Nach dem ein Client mit dem pvsmgr eine Verbindung erfolgreich aufgebaut hat, wird er im ConnectionWindow als ein richtiger VNC-Frame angezeigt. Der *Nicht dummy-Frame* hat einen grünen Punkt an seine rechte Ecke. Der Manager kann also

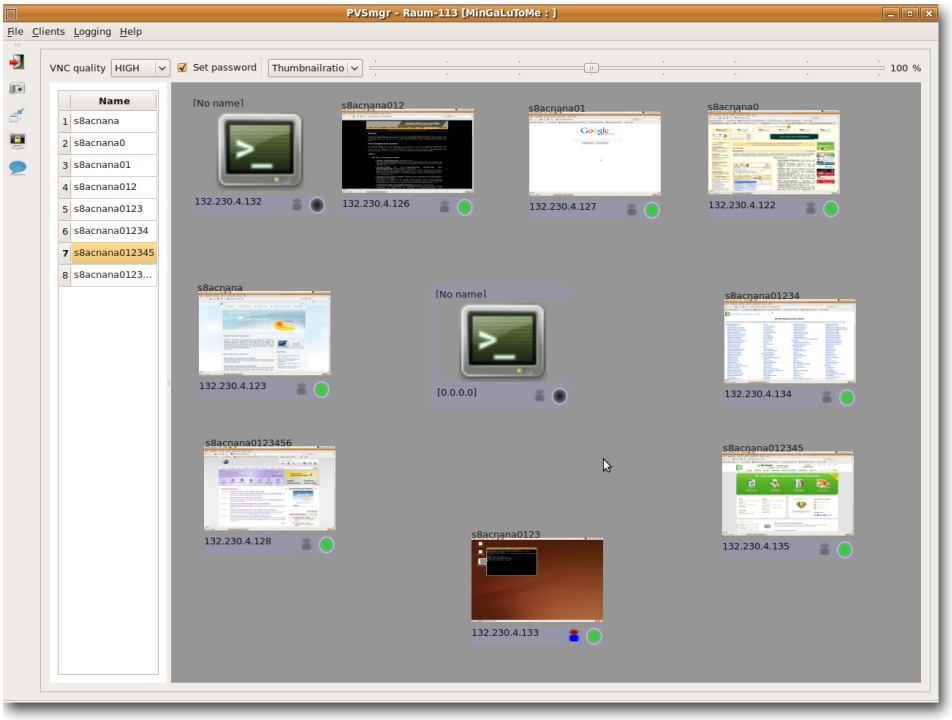

Abbildung 4.2: Dummy und Nicht Dummy Frames

diese Frame manipulieren. Dem Dozent steht dafür mehrere Funktionen zur Verfügung. Wir werden aber hier nur die wichtigsten erläutern, denn die anderen lassen sich ganz trivial auf dem Gui nachvollziehen. Also auf ein VNC-Frame kann per Rechtsmouseklick die im 4.1.1 definerten Aktionen angewandt werden. Die VNC-Qualität und die Größe der Frames können geändert werden, um den CPU-Last zu minimieren. Auf dem Bild 4.2 kann man verbundene Clients (*Nicht dummy-Frame*) und nicht verbundene Clients (*Dummy-Frame*) ansehen . Der Titel des Fensters ist der Benutzername des Clients und unter dem Frame befindet sich die korrespondierte IP-Adresse. Der grüne Punkt ganz rechts im Gegenteil zum dummy-Frame zeigt, dass der Client gerade Online ist.
Für die Verbundene Clients besteht die Möglichkeit die VNC-Quality (HIGH, MEDIUM oder LOW) jeder Zeit einzustellen. Mit *Set Password* kann der Manager ganz bequem festlegen, ob die Clients zur Verbindung mit dem pvsmgr ein von System generiertes Passwort eingeben müssen oder nicht. Auf dem Toolbar werden weitere Funktionen wie: Screeshots aufnehmen, Chat mit dem einzelnen Client starten oder alle Clients auf einem Klick zu blockieren. Das Blockieren der Clients schließt natürlich der Dozent-Rechner aus. Der Dozent-Rechner hebt sich von einem anderer verbundenen Machine durch den Rot-Blau Zeichnen neben dem grünen Punkt hervor.

Im letzten Punkt dieser Abschnitt wollen wir die vorhandenen Tastenkürzel vorstellen.

4.1.3 Tastenkürzel für PVS-Server

Einige schlaue Tastenkombinationen sollen die Ausnutzung der PVS-Konsole erleichten. Wir werden z.B für einige Funktionen, die in der Popup-Menü zur Verfügung gestellten sind, den Aufruf durch Tastenkombination ermöglichen.

Wichtig ist, eine Rückkopplung mit den Anwendern vorzunehmen (Rückmeldung, was die Dozenten denken und welche Funktionen sie für sinnvoll erachten, was sie sich wünschen ...)

Hier folgt eine Liste von den bisherigen eingebauten Tastenkombinationen:
Alt+F: Anzeige des Menüs unter "File"
Alt+C: Anzeige des Menüs unter "Clients"
Alt+L: Anzeige des Menüs unter "Logging"
Alt+H: Anzeige des Menüs unter "Help"
Ctrl+M: Anzeige des Profilmanager-Dialogs
Ctrl+Q: Die Anwendung verlassen (Exit)
Ctrl+1: Name in Client-Liste anzeigen lassen
Ctrl+2: IP in Client-Liste anzeigen lassen
Ctrl+3: Username in Client-Liste anzeigen lassen
Ctrl+L: Log anzeigen lassen
Ctrl+O: Normal Log anzeigen lassen
Ctrl+R: Error Log anzeigen lassen
Ctrl+N: Network Log anzeigen lassen
Ctrl+T: Terminal Log anzeigen lassen
Ctrl+D: Chat Log anzeigen lassen
Ctrl+F: Screenshort der ausgewählten Clients machen
Ctrl+V: Ausgewählter Client in voller Auflösung anzeigen lassen
Ctrl+A: Alle Clients Un- bzw. Lock

5 PVS-Client

Eine der Anforderungen an die Software ist, dass sie den Benutzer nicht an der Arbeit hindert und im Hintergrund laufen soll. Dies wurde mit Hilfe einer Applikation realisiert, die nur aus einer Toolbar besteht und bei Nichtgebrauch automatisch ausgeblendet wird. Falls der aktuelle Windowmanager dies unterstützt, wird zusätzlich ein Symbol (schwarze Kamera) im Systemabschnitt der Taskleiste (System Tray) abgelegt. In diesem Fall, werden dem Benutzer bei diversen Ereignissen Meldungen über sogenannte „Sprechblasen“ angezeigt. Wenn der Windowmanager keinen Systemabschnitt besitzt, werden diese Meldungen über konventionelle Dialoge präsentiert. Des Weiteren informiert das Symbol über den Verbindungsstatus. Im Falle einer erfolgreichen Verbindung zu einer Steuerkonsole, wechselt die kleine Lampe von rot auf grün.
Eine weitere Anforderung ist der ständige Zugriff auf die Benutzerschnittstelle damit zu jeder Zeit Einstellungen vorgenommen werden können. Dies wird ebenfalls gewährleistet wenn ein Video oder eine virtuelle Maschine im Vollbildmodus dargestellt wird. Des Weiteren kann die Toolbar entweder am oberen oder unteren Bildschirmrand an einer variablen waagerechten Position angeordnet werden. Dies ist deshalb wichtig, da diverse Windowmanager ihre eigenen Toolbars an unterschiedlichen Orten platzieren.
Die Toolbar des PVS-Clients (Abb. 5.1) besteht aus einer Reihe von Bedienelementen, die nachfolgend erklärt werden:

1. Die wichtigste Schaltfläche der Toolbar – ein Klick auf diese öffnet das Hauptmenü. Das Menü bietet Zugang zu allen Funktionen die dem Benutzer zur Verfügung stehen. Zusätzlich ist dieses Menü über einen Rechtsklick auf das Symbol im Systemabschnitt der Taskleiste (System Tray) abrufbar.

2. Die Schaltfläche links neben der Beschriftung „Host“ erlaubt eine schnelle Wahl der Steuerkonsole. Durch einen Klick auf diese öffnet sich ein Menü mit einer Auswahl der verfügbaren Server. Durch einen weiteren Klick auf einen der angezeigten Namen kann nach der Eingabe eines Passwortes zu der gewählten Steuerkonsole verbunden werden. Falls kein Passwort benötigt wird, kann das Feld leer bleiben.
   Die Beschriftung dieser Schaltfläche wird nach einer erfolgreichen Verbindung an den Namen der Steuerkonsole angepasst, wobei ein „-“ signalisiert, dass keine Verbindung besteht. Die Liste möglicher Server wird zur Laufzeit automatisch aktualisiert.

3. Diese Checkbox erlaubt es dem Benutzer zu bestimmen ob die verbundene Steuerkonsole das Recht hat eine VNC-Verbindung zum Client aufzubauen. Falls aktiviert, kann

der Benutzer der die Steuerkonsole gestartet hat das Verhalten des Benutzers am Client beobachten und bei Bedarf dessen Bild im Netzwerk verteilen.

Abbildung 5.1: Toolbar des PVS-Clients

Das Hauptmenü des PVS-Clients besteht aus folgenden Elementen:

- Connect: Zu einer Steuerkonsole verbinden. Der Vorgang ist dabei analog zur Schnellwahl in der Toolbar.
- Disconnect: Eine bestehende Verbindung trennen.
- Information: Zeigt in einem Dialog den Namen der aktuell verbundenen Steuerkonsole sowie das zugehörige Passwort. Diese Funktion ist für Dozenten nützlich, um beispielsweise über einen angeschlossenen Projektor die Daten für alle Clients bekannt zu geben.
- Chat: Öffnet den Chat-Dialog.
- Send File: Ermöglicht das Senden einer Datei an einen Teilnehmer.
- Config: Öffnet den Konfigurationsdialog.
- About: Zeigt Versionsinformationen an.
- Quit: Beendet die Applikation sowie das zugehörige Backend.

5.1 Chat

Abbildung 5.2 zeigt das Chatfenster des PVS-Clients. Mit diesem ist es möglich mit anderen Teilnehmern gemeinsam zu kommunizieren oder private Gespräche zwischen zwei Teilnehmern zu führen.

1. Dies ist eine Liste aller aktuell am Chat teilnehmenden Benutzer. Ein Doppelklick auf einen Namen ermöglicht ein privates Gespräch, mit einem Rechtsklick kann dem gewählten Benutzer eine Datei geschickt werden.
2. Das Hauptfenster des Dialogs zeigt den aktuellen Gesprächsverlauf an. Dabei wird jeder Nachricht die Uhrzeit sowie der Name des Absenders hinzugefügt. Hier erscheinen ebenfalls diverse Ereignisse, wie der Beitritt eines neuen Benutzers oder der Wechsel des Verbindungsstatus.
3. Durch diese Reiter kann das aktuelle Hauptfenster und somit der Gesprächspartner gewechselt werden. Private Unterhaltungen können durch das kleine „x“ geschlossen werden. Bei neuen, also noch nicht gelesenen Nachrichten, erscheint eine grüne Lampe.

4. Hier wird die eigene Nachricht eingegeben.

5. Ein Klick auf „Send“ verschickt die eingegebene Nachricht entweder an den öffentlichen Kanal oder bei einer privaten Unterhaltung direkt an den Kommunikationspartner. Der Empfänger ist dabei stets derjenige dessen Gesprächsverlauf im Hauptfenster angezeigt wird. Es ist auch möglich nach der Eingabe einer Nachricht zum Abschicken die Enter-Taste zu betätigen.

Das Chatfenster unterstützt ebenfalls „Drag & Drop“. Das Ablegen einer Datei innerhalb des Fensters bewirkt eine Anfrage beim aktuellen Gesprächspartner zur Dateiübertragung.
Sollte eine Nachricht eingegangen sein obwohl der Chat-Dialog geschlossen ist, so wird dies dem Benutzer über eine „Sprechblase“ im Systemabschnitt der Taskleiste mitgeteilt.

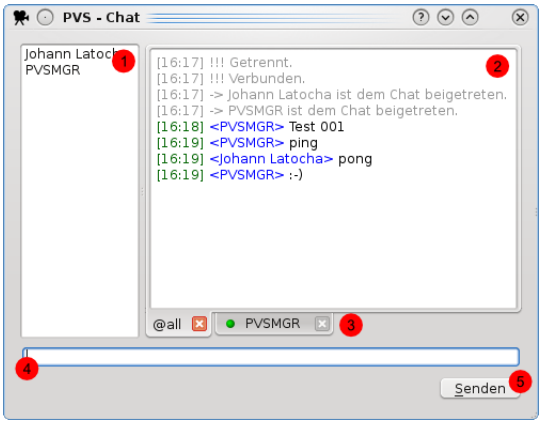

Abbildung 5.2: Chat-Dialog des PVS-Clients

5.2 Konfiguration

Der Konfigurationsdialog (Abb. 5.3) erlaubt es dem Benutzer des PVS-Clients diverse Einstellungen vornehmen:

1. Mit diesen Reitern kann die Gruppe der zur Verfügung stehenden Optionen gewechselt werden. Unter „Permissions“ sind Einstellungen zu finden mit denen der Benutzer seine Privatsphäre schützen kann. Unter „Display“ ist es möglich das Verhalten der Toolbar anzupassen (z.B. die Position selbiger).

2. Hier wird dem Dozenten das Recht erteilt eine VNC-Verbindung aufbauen zu können (z.Z ohne Funktion).

3. Hier wird allen anderen Benutzern das Recht erteilt eine VNC-Verbindung aufbauen zu können (z.Z ohne Funktion).

4. Durch diese zwei Checkboxen kann die Teilnahme am Chat oder das Akzeptieren von Dateiübertragungen aktiviert werden (z.Z ohne Funktion).

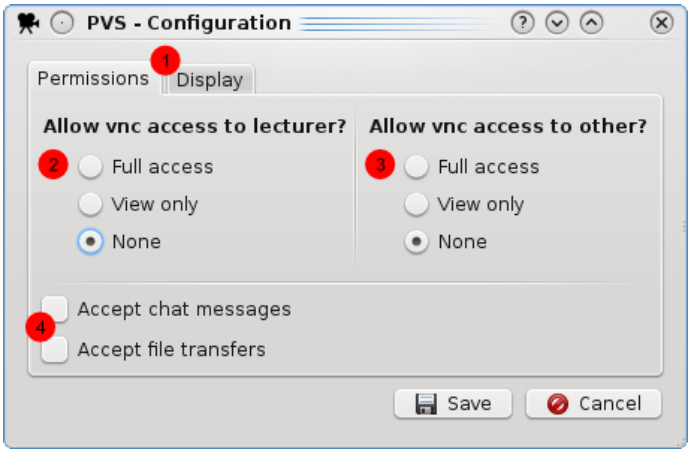

Abbildung 5.3: Konfigurationsdialog des PVS-Clients

Teil III

Entwicklerdokumentation

6 Erzeugen und Installieren der Applikation

Um den Pool Video Switch erfolgreich zu kompilieren, müssen zunächst die Quelldateien heruntergeladen werden. Hierzu sollte auf dem Arbeitsrechner ein Subversion-Client installiert sein. Ein Checkout kann in einem beliebigen Terminal erfolgen, wobei es zwei unterschiedliche Arten des Zugriffs gibt.

Anonym, nur Leserechte:
svn co http://svn.openslx.org/svn/pvs/trunk/pvs

Mit Account, auch Schreibrechte (benutzer mit SSH Login ersetzen):
svn co svn+ssh://benutzer@openslx.org/srv/svn/pvs

6.1 Voraussetzungen

Der Pool Video Switch benötigt zum Kompilieren folgende Pakete (und ihre Abhängigkeiten):

- libvncserver-dev >= 0.9.3
  Eine angenehm kleine, wenn auch in C geschriebene, Bibliothek für die Erstellung von VNC Servern und Clients.

- libx11-dev >= 1.3.3
  Diese Bibliothek wurde eingesetzt, um ausgewählten oder allen Zielrechnern den Zugriff auf Maus und Tastatur zu entziehen und den Bildschirm schwarz zu schalten.

- libqt4-dev >= 4.5.3
  Sowohl die Server als auch die ClientGUI benutzen das Qt Framework in der Version 4. Ferner wird QtNetwork zur Kommunikation zwischen Server und Client benötigt und QtDbus zur Interprozesskommunikation zwischen Client-Daemon und GUI.

- qt4-dev-tools >= 4.5.3
  Dies wird nur benötigt, falls Unterstützung weiterer Sprachen implementiert werden soll, da in diesem Paket das Hilfsprogramm **linguist** enthalten ist.

- cmake >= 2.4.0
  Eingesetztes Buildsystem und der Makefile-Generator. Um distributionsspezifische Pakete zu erzeugen wird CPack benutzt.

6.2 Kompilieren aus Quellen

Sobald alle Voraussetzungen erfüllt und die Quellen installiert sind, kann das ausführbare Programm erzeugt werden. Falls gewünscht, kann zusätzlich in der Datei *CMakeLists.txt* der Build-Type eingestellt werden. Gültige Werte sind `Debug` und `Release` und ihre Wirkung den zwei weiteren Zeilen zu entnehmen.

```
SET(CMAKE_BUILD_TYPE Debug)
SET(CMAKE_CXX_FLAGS_DEBUG "-O0 -g -Wall")
SET(CMAKE_CXX_FLAGS_RELEASE "-O3 -march=native")
```

Damit das Wurzelverzeichnis des Projekts weiterhin sauber bleibt (Übersicht beibehalten, Arbeit mit SVN erleichtern), werden wir hier einen Out-Of-Source Build durchführen. Dies bedeutet, dass zum Kompilieren ein seperates Verzeichnis benutzt wird und sämtliche automatisch generierten Dateien sowie das Kompilat selbst hier abgelegt wird. In einem Terminal sollte nun in das Verzeichnis indem sich das Projekt befindet gewechselt werden, um folgende Befehle ausführen zu können:

mkdir -p build
cd build/
cmake ..
make

Die verschiedenen Applikationen können nun folgendermaßen (in *build/*) ausgeführt und getestet werden:

- Den Server starten: **./pvsmgr**
- Den Server mit Touchscreenoberfläche starten: **./pvsmgrtouch**
- Den ClientDaemon starten: **./pvs**
- Die ClientGUI starten: **./pvsgui**

6.3 Installation

Nachdem das Projekt kompiliert wurde, kann es (als Superuser z.B. root) auf dem lokalen System installiert werden:
make install

Dabei werden folgende Dateien auf das aktuelle System übertragen:
/usr/local/bin/pvsmgr

/usr/local/bin/pvs
/usr/local/bin/pvsgui
/usr/local/bin/pvsmgrtouch
/usr/local/share/dbus-1/services/org.openslx.pvs.service

Ein Vorteil ist, dass nun ein seperates Ausführen des Client-Daemon nicht mehr notwendig ist. Dieser wird ab sofort automatisch gestartet und beendet, sobald die ClientGUI ausgeführt wurde.

Es ist ebenfalls möglich, distributionsspezifische Pakete zu erstellen (zur Zeit nur .deb):
make package

Falls erfolgreich, befindet sich nun eine Installationsdatei (pvs-<version>-Linux.deb) in *build/* die z.B. auf Debian-Derivaten mit **dpkg -i** installiert werden kann.
Anmerkung: Falls Pakete für Rechner gebaut werden sollen, die vom Erstellungssystem her verschieden sind, muß der Schalter `-march=native` in der Datei *CMakeLists.txt* angepasst werden!

7 Eingesetzte GUI-Bibliothek

Die komplette Benutzerschnittstelle des Clients wurde mit Hilfe des von Nokia (früher Trolltech) entwickelten Frameworks Qt in der Version 4 implementiert. Der Grund hierfür sind folgende Vorteile:

- Qt wird bereits in einer Vielzahl von großen Projekten eingesetzt (KDE, Opera, Skype, Google Earth,...) und ist somit solide und erprobt.
- Das komplette Framework besteht nicht nur aus einer GUI-Bibliothek sondern beinhaltet noch viele weitere nützliche Module, die bei der Entwicklung ebenfalls zum Einsatz kamen (QtNetwork, QtDBus, Internationalisierung).
- Qt ist plattformunabhängig: Projekte können für Linux, OSX und Windows erstellt werden
- Moderne Konzepte und Werkzeuge werden bereitgestellt. So wird das Signal-Slot-Konzept durchgängig eingehalten und mit dem GUI-Builder (**designer**) können einfach und professionell grafische Interfaces erstellt werden.
- Duales Lizenzmodell: Qt gibt es sowohl unter einer proprietären Lizenz als auch unter der (L)GPL.

Leider unterstützte das bisher eingesetzte Build System (Autotools) Qt nicht vollständig. Dies liegt daran dass Qt zum Erzeugen von Makefiles ein eigenes Werkzeug mitbringt (QMake). Aus diesem Grund wurde eine Umstellung auf das neue CMake durchgeführt. Dies ist nicht nur flexibler als Autotools (plattformunabhängig) sondern auch einfacher in der Einrichtung und Wartung (und wird übrigens vom KDE-Projekt seit der Version 4 eingesetzt). Des Weiteren ist es mit CMake nicht nur möglich Qt-Anwendungen zu erstellen sondern auch die komplette Qt-Toolchain (**moc**, **uic**, **rcc**, **qdbuscpp2xml**, **qdbusxml2cpp**, **lupdate**, **lrelease**,...) komfortabel zu nutzen. So kann z.B. die Interface Erstellung für D-Bus oder das Internationalisieren der Software automatisiert werden.

7.1 Internationalisierung

Wie oben bereits erwähnt wurde zur Internationalisierung das von Qt4 bereitgestellte System benutzt und mit CMake automatisiert. Der interne Ablauf sieht folgendermaßen aus (*CMakeLists.txt*):

1. Mit `SET` werden Variablen mit Quelldateien definiert, die die zu übersetzenden Zeichenketten beinhalten. Diese dienen gleichzeitig zum Kompilieren.
2. Danach werden ebenfalls mit `SET` Variablen definiert, die später die vom Benutzer zu übersetzenden .ts Dateien enthalten (z.B. *i18n/pvsgui_de_DE.ts*).
3. `QT4_CREATE_TRANSLATION` erzeugt im Build-Verzeichnis binäre Internationalisierungs Dateien, die beim Kompilieren eingebunden werden müssen.

Beispiel:

```
SET( PVSGUI_SRCS src/pvsgui.cpp src/gui/clientConfigDialog.cpp )
SET( PVSGUI_TSS i18n/pvsgui_de_DE.ts i18n/pvsgui_fr_FR.ts )
QT4_CREATE_TRANSLATION( PVSGUI_QMS ${PVSGUI_SRCS} ${PVSGUI_TSS} )
ADD_EXECUTABLE( pvsgui ${PVSGUI_SRCS} ${PVSGUI_QMS} )
```

Jede so erzeugte .ts Datei kann nun mit dem Programm **linguist** von einem Übersetzer bearbeitet werden. Damit die Zeichenketten später während der Ausführung übersetzt werden, müssen diese allerdings mit der Funktion `QObject::tr()` im Quellcode erzeugt worden sein.

8 Aufbau und Funktionsweise des PVS

Generelles Konzept

8.1 Einzelne Komponenten

8.1.1 Zuordnung von Konsole und Clients

Die Zuordnung von Clients zur Konsole erfolgt halb automatisch über ein Service-Discovery-Modul. Jede Konsole sendet im Intervall von 7 Sekunden einen Broadcast ins lokale Subnetz, um ihre Verfügbarkeit bekannt zu geben. Die Clients können dadurch eine Liste von verfügbaren Servern anzeigen, aus der der Benutzer nur noch den gewünschten auswählen muss. Das Intervall sowie die verwendeten UDP-Ports für das Service-Discovery können vor dem Kompilieren in der *setup.h* festgelegt werden

```
#define SB_INTERVAL 7 // Broadcast interval in seconds
#define SD_PORT_CONSOLE 3491 // Not really used, Qt just wants a bind
#define SD_PORT_CLIENT 3492 // This is where we expect announcements
```

Die zuständigen Klassen sind *PVSServiceBroadcast* (Serverseitig) und *PVSServiceDiscovery* (Clientseitig).
Auf Seite des Clients erfolgt außerdem eine Kommunikation mit der GUI über DBus, um verfügbare Sitzungen anzeigen zu lassen, und damit die GUI den Daemon veranlassen kann, zu einer bestimmten Sitzung zu verbinden. Die Methode *connectToSession* in *PVSServiceDiscovery* ist dafür zuständig, zu einem gegebenem Sitzungsnamen die nötigen Daten zu ermitteln und eine Verbindung zu initialisieren.

Absicherung und Verifikation der Identität

Um das Fälschen bzw. Manipulieren von diesen Broadcasts zu verhindern, sodass es nicht möglich ist, Man-in-the-Middle Angriffe oder ähnliches durchzuführen, soll gewährleistet werden, dass eine angezeigte Sitzung auch die ist, für die man sie hält.
Dafür wird auf das SSL-Protokoll zurückgegriffen. Zum einen bietet SSL Verschlüsselung für Verbindungen an, und das Qt-Framework hat hierfür auch schon einige Klassen parat. Zum anderen ist mit SSL durch seine Zertifikate auch Identitätsverifikation möglich. Dazu kann das

Zertifikat des Servers anhand des Fingerprints auf Echtheit überprüft werden. Beim Start der Konsole wird also, falls noch nicht vorhanden, ein neues Serverzertifikat inklusive Schlüsselpaar erstellt, mit welchem Verbindungen mit Clients fortan verschlüsselt werden. Außerdem wird der Fingerprint des Zertifikats in den Service-Broadcasts übertragen, sodass die Clients bereits vor der Verbindung den zu erwartenden Fingerprint kennen. Somit kann beim Aufbau der Verbindung zur Konsole der Fingerprint verglichen und eine eventuelle Manipulation erkannt werden.
Für die Eindeutigkeit ist nur noch eine feste Zuordnung von Sitzungsnamen, welcher dem Benutzer angezeigt wird, zum Fingerprint der zugehörigen Konsole notwendig. Dies wird mit Hilfe eines einfachen Algorithmus erreicht, der aus dem Fingerprint einen lesbaren String erzeugt. Wichtig ist, dass Konsole und Client hier denselben Algorithmus verwenden, damit der Dozent an der Konsole den Studenten den korrekten Sitzungsnamen mitteilen kann. Wird der Algorithmus auf Seite der Konsole verändert, müssen auch alle Clients aktualisiert werden. Er befindet sich in der *serviceDiscoveryUtil.h* und heißt *sha1ToReadable*.
Empfängt der Client einen Broadcast über einen neuen Server, baut dieser zunächst zur Überprüfung eine Verbindung zu diesem Server auf und sofort wieder ab, um den Fingerprint überprüfen zu können. Nur wenn dieser mit dem angekündigten Fingerprint aus dem Broadcast übereinstimmt, wird die Sitzung dem Benutzer zur Auswahl angezeigt.
Die Überprüfung geschieht nicht sofort beim Empfang eines Service-Announcements sondern verzögert in einem Timer, und dabei auch jeweils nur ein Server in jedem Timer-Aufruf. Dies soll Missbrauch durch gefälschte Broadcastpakete verhindern oder zumindest stark verlangsamen. Entsprechende Mechanismen gegen Missbrauch finden sich in *PVSServiceDiscovery::handleDiscovery* sowie *PVSServiceDiscovery::timerEvent*.

Generierung von Sitzungsnamen

Hier ist prinzipiell viel Spielraum für Kreativität vorhanden, beachtet werden sollte lediglich, dass die Zahl der möglichen Namen, die durch den Algorithmus erzeugt werden können, hoch genug ist, um nicht in kurzer Zeit durch einen Brute-Force-Angriff einen Fingerprint mit dem selben Sitzungsnamen erzeugen zu können. Außerdem sollte die Verteilung der zufälligen Namen über den Raum der möglichen Namen etwa gleichmäßig sein.
Der momentan verwendete Algorithmus generiert fünfsilbige Wörter, wobei jede Silbe aus einem Konsonanten und einem Vokal besteht. Die Konsonanten werden aus einer Liste von 13 Stück ausgewählt, wobei identisch oder zu ähnlich klingende Konsonanten herausgefiltert wurden, um bei der mündlichen Übertragung keine Ambiguitäten zu erzeugen. mit einer Wahrscheinlichkeit von 25% wird einer Silbe ein **n** angehängt. Damit sind $130^5 = 37.129.300.000$ (37 Mrd.) unterschiedliche Sitzungsnamen möglich, was zwar deutlich weniger sind, als bei einem 160Bit langen SHA1-Hash, allerdings ist dies bezogen auf das Einsatzgebiet mehr als ausreichend. Alternativ bleibt natürlich nach wie vor das verwenden alternativer Algorithmen die einen größeren Raum an möglichen Namen haben.

8.2 Überblick über Aktivitäten auf Clients

8.2.1 Projektion an Alle

Eine wichtige Eigenschaft des PVS ist die Verwaltung von Projektionen zwischen mehreren Clients. Eine Projektion ist hierbei das Anzeigen des Bildschirminhalts eines Clients - der sogenannten Source oder Quelle - auf einem oder mehreren anderen Clients - den Targets oder Zielen der Projektion. Die für die Projektion benötigten Verbindungsdaten wie Passwort und IP werden von jedem Client bei der Anmeldung an der Steuerkonsole übermittelt und in einer Liste von PVSConnection Objekten in der Klasse *PVSConnectionManager* gespeichert. Diese zentrale Verwaltung hat mehrere Vorteile:

- Die Quelle einer Projektion muss keine Aktion ausführen und kann passiv bleiben.
- Redundanz der Daten wird verhindert, da diese auch in der Steuerkonsole zur Darstellung der Thumbnails benötigt werden.
- Das Nachrichtenaufkommen wird reduziert, da lediglich eine Nachricht bei der Anmeldung an der Steuerkonsole übermittelt wird.

Bei der Auswahl der Quelle und Ziele ist zu beachten, dass man für jede Projektion jeweils nur eine Quelle jedoch mehrere Ziele auswählen kann. Quelle und Ziel müssen außerdem verschiedenen Kriterien genügen.

- Eine Quelle darf nicht gleichzeitig Ziel einer Projektion sein.
- Ein Ziel einer Projektion darf nicht Ziel einer anderen Projektion sein.
- Eine Quelle darf mehrfach als Quelle ausgewählt werden.

Diese Einschränkungen werden in der Steuerkonsole durchgesetzt, indem im Zielauswahldialog die Zielmenge eingeschränkt wird. Siehe hierzu auch 9.3.1 Projektion.
Der Projektionsvorgang an sich besteht aus mehreren Teilen. Wird eine Projektion angefordert, wird überprüft, ob auf der Quelle ein VNC Server gestartet ist. Falls nicht, wird versucht, einen VNC Server zu starten. Ist dies erfolgreich, so sendet die Steuerkonsole das entsprechende Tripel (IP, Passwort und Port der Quelle) an alle ausgewählten Ziele. Clients, welche eine Projektionsaufforderung erhalten, verbinden sich dann mit den Verbindungsdaten zum VNC Server der Quelle. Um die Einstellbarkeit der Qualität einer Projektion zu ermöglichen, kann die Steuerkonsole einen von drei Qualitätswerten an die Zielclients übermitteln. Siehe hierzu auch 9.3.1 Qualitätsoptionen.

8.2.2 Projektion eines Clients auf dem Beamer

Die Projektion eines Clients an den Beamer unterscheidet sich im Wesentlichen nicht von anderen Projektionen. Lediglich ist das Ziel der Projektion hierbei der Dozentenpc bzw. der PC, welcher an den Beamer angeschlossen ist. Eine spezielle Auszeichnung des Beamers erfolgt nicht. Die Anzahl der Ziele wird hierbei nicht beschränkt, da es wünschenswert sein kann, den auf dem Beamer dargestellten Bildschirminhalt auch gleichzeitig auf anderen Clients darzustellen.

8.2.3 Chat- und Informationskanal

Es gibt 2 Möglichkeiten um Kontakt mit den Clients aufzunehmen. Die erste ist über den Chat, wo Nachrichten sowohl über den offentlichen Kanal als auch über einen privaten Kanäle verteilt werden können, und die zweite vom PVSManager aus über den Informationskanal. Der Informationskanal ermöglich das Versenden von Nachrichten, die dringend zu lesen sind, an die Clients. Im Gegenteil zum Chat erscheinen solche Nachrichten nicht im Chatfenster sondern in einem Pop-Up-Fenster und werden vom Bildschirm entfernt erst wenn man sie als gelesen markiert durch das Drucken auf dem Knopf *'OK'*.

Behandlung der Nachrichten im Server

Chat-Nachtichten werden von Server in der Klasse `PVSConnectionManager` mittels der Methode `onChat` behandelt. Dort wird aus der Nachrit den Empfänger und den Absender ausglesen und die Nachricht an beide versendet. So gilt das Empfangen eine eigene Nachricht als Bestätigung, dass die Nachricht ordentlich vom Server behandelt und versendet wurde. Das Gestalt von solchen Nachrichten sieht folgendermaßen aus

PVSMSG

Type	Ident	Msg	sndID
`PVSMESSAGE`	`<Username des Empfängers>`	`«Username des Absenders>:<Die eigentliche nachrich>`	`«Vom server zugewissene ID des Absenders>`

Informationsnachrichten werden ausschließlich vom PVSManager versendet.Dies geschiet in der Klasse `ConnectionList` mittels der Methode `on_Message`.

PVSMSG

Type	Ident	Msg
`PVSMESSAGE`	`BROADCAST`	`<Die eigentliche nachrich>`

Informationnachrichten können außerdem einen oder mehrere Clients sperren, wenn sie den Ident `LOCKSTATION` enthalten. Sobald ein Client die Nachricht empfängt, wird diese auf dem Bilschirm angezeigt und 10 Sekunden später wird der Client gesperrt.
Abgesehen von der Behandlung der Nachrichten muss sich der Server darum kümmern, dass jeder verbunde Client über alle nötige Informationionen verfügt damit er Nachrichten mit andren Clients austauschen kann. Dies wird folgendermaßen erledigt:

- **Einfügen eines Clients:** um die Verwaltung von Clients kümmert sich die Klasse `PVSConnectionManager`, in der die Methode `onClientNew` für das Einfügen von neuen Clients zuständigt ist. Sowald ein neuer Client in der Client-Liste des Servers eingefügt wird, wird an ihn die Liste aller im Server bereits angemeldete Clients geschickt. Dazu dient die Methode `sendEventToClients`.

Bis hier ist der neue Client noch unbekannt für den Rest der Nutzer. Der neuer Client wird erst bekannt gegeben sobald er vom Server einen Benutzername zugewissen bekommen hat. Da es sein kann, dass den Name, mit dem der neue Client sich beim Server anmelden wollte, bereits vergeben ist und muss unter Umständen verändert werden. Diese Zuweisung findet in der Methode `onLoginUsername` statt, wo nicht nur alle andere Clients sondern auch der neue Client darüber informiert werden. Auch hier kümmert sich die Methoder `sendEventToClients` ums Vesenden der entsprechenden Informationen.

- **Entfernen eines Clients:** das Entfernen von Clients wird von der Methode `onClientRemove` erledigt, wo analog wie Oben alle Clients darüber informiert werden.

Für dei Übermittlung solche Informationen werden Nachrichten mit folgenden Gestal benutzt

PVSMSG

Type	Ident	Msg
PVSMESSAGE	<Befehl>	<Benutzername>:<IP-Adresse>

Es gibt drei unterschiedliche Befehle, die welche Änderung in der localen Client-Liste der Clients vorgenommen werden soll angeben.

1. `clientToAdd`
2. `clientToRemove`
3. `assignedName`

Wie es bei Servern gewöhnt ist, werden alle relevante Ereignisse in Log-Dateien protokolliert. Ereignisse werden im Chat-Log mit folgendem Befehl eingetragen

```
ConsoleLog writeChat(<Beschreibung des Ereignisses>)
```

Chat-Interface der Steuerkonsole

So wie alle Clients ist der PVSManager auch ein Teilnehmer im Chat. Der PVSManager steht wie alle andere Teilnehmer auch in der Nutzer-Liste jedes Chat-Fenster und kann ebenfalls über private Nachrichten direkt angesprochen werden. Die Arbeitsweise dieser Chat-Interface ist sehr simple. Da sie sich im Server befindet, müssen einfach alle Ereignise (Nachrichten senden ist die einzige Ausnahme) von der Klasse `PVSConnectionManager` an die Klasse `MainWindow` weitergegeben werden. Dies kümmert sich darum, alle Informationen zu verarbeiten und diesein der Chat-Fenster der Steuerkonsole anzuzeigen.
Folgende Methoden der Klasse `MainWindow` ermöglichen das Anzeigen einer empfangenen Nachricht im Chat-Fenster der Steuerkonsole und Änderungen (Clients einfügen und entfernen) in der sowohl im Chat-Fenster als auch in der Steuerkonsole angezeigten Client-Liste.

```
receiveChatMsg(<Absender>, <Empfänger>, <Nachricht>)
removeConnection(*pvsClient)
```

`addConnection(*pvsClient)`

Alle diese Methoden werden im Gegensatz von der Methode `sendChatMsg(PVSMsg myMsg)` von der Klasse `PVSConnectionManager` aus ausgeführt. Da alle durchs Netz empfangene Nachrichten müssen an die GUI-Weitergegeben werden. Beim Versenden von Nachrichten funktioniert es genau umgekehrt. Die Nachricht wird vom Nutzer in der GUI eingegeben und muss an die Klasse `PVSConnectionManager` weitergeleitet werden, damit diese ins Netz gesendet wird. Darum kümmert sich die Methode in der Klasse `MainWindow`.

`MainWindow::sendChatMsg(PVSMsg myMsg)`

Chat-Clients

So weit haben wir die Funtionsweisen des Servers im Bezug auf dem Chat kennengelernt. Nun wird erläutert wie die einzelnen Clients die Nachrichten bearbeiten.

Auf der Client-Seite in der Klasse `PVSConnectionServer` werden alle Nachrichten des PVS-Protokolls empfangen und gefiltert (Siehe Methode `handleClientMsg`). Nachrichten mit dem Ident `PVSMESSAGE` werden durch den Dispatcher direkt an die Klasse `PVSChatClient` weitergeleitet, wo die Methode `receive` feststellen wird, ob es sich dabei um eine Gespräch-Nachricht oder eine Befehl-Nachricht handelt. Um es feststellen zu können, wird aus der empfangenen Nachricht ein PVSChatMsg-Objekt erzeugt (siehe Konstruktor) und mittels der Methode `isCommand` erfährt man ob es sich um einen Befehl handelt oder nicht. Falls ja leitet der Dispatche die Nachricht an die Stelle `PVS::UpdateChatClients` sonst an die Stelle `PVS::chat_receive`, wo die Änderungen in der Client-Liste vorgenommen werden oder die Gespräch-Nachricht der GUI abgegeben wird.

8.3 Netzwerkkommunikation

8.3.1 PVS-Protokoll

Im Zuge der Entwicklung des PVS wurde ein sehr einfaches Messagingprotokoll entwickelt. Die Nachrichten bzw. Messages bestehen dabei aus Header, Nachrichtentyp, Ident und dem eigentlichen Nachrichtentext. Die einzelnen Nachrichtenteile, welche bis auf den Header selbst definiert werden können, werden verknüpft und versendet. Es sind schon Nachrichtentypen vordefiniert, welche sich nur durch unterschiedliche Dispatcher unterscheiden. Bereits vorhandene Typen sind COMMAND, LOGIN, MESSAGE und UNKNOWN. Die Dispatcher (_commandDispatcher, _loginDispatcher und _chatDispatcher) befinden sich im Client in der Klasse `ServerConnection`, in der Steuerkonsole in der Klasse `ClientConnection` bzw. `ListenServer`. Ein Ident wie z.B. Username, Befehl oder Beschreibung des Nachrichteninhalts dient zur Unterscheidung verschiedener Nachrichten mit demselben Nachrichtentyp, wobei die Nachricht dann den dem Ident entsprechenden Nachrichtentext enthält. Um eine Funktion zur Behandlung einer bestimmten Nachricht zu definieren, wird diese Funktion als Handler mit dem entsprechenden Dispatcher verknüpft. Im PVS-Client beispielsweise befindet

sich der Dispatcher für Nachrichten vom Typ Command in der Klasse `pvsServerConnection`, daher wird in der Klasse `pvs` die Funktion `void PVS::onCommand(PVSMsg cmdMessage)` wie folgt als Handler registriert: `_pvsServerConnection->addCommandHandler("*", this, &PVS::onCommand)`. Erhält nun der Client eine Nachricht vom Typ COMMAND, so wird die Funktion onCommand mit dem entsprechenden Nachrichtenobjekt aufgerufen. Auf die eigentliche Nachricht kann dann über die Methoden `getIdent()` und `getMessage()` des Objektes zugegriffen werden.

8.3.2 PVS-Messages

In Tabelle 8.1 sind die Messages, die zwischen den einzelnen PVS Komponenenten ausgetauscht werden, aufgelistet. Der Nachrichtentyp gibt dabei an, welcher Dispatcher die Nachricht behandelt. Der Ident einer Nachricht wird zur Verarbeitung einer empfangenen Nachricht in der aufzurufenden Funktion benötigt (Unterscheidung von anderen Nachrichten gleichen Typs).

Nachrichtentyp	Nachrichten Ident	Inhalt	Beschreibung
PVSCOMMAND	PROJECT	hostname port password quality	Hostname, Port, Passwort und Qualität des VNC Servers zu dem verbunden werden soll (durch Space getrennt)
PVSCOMMAND	UNPROJECT		
PVSCOMMAND	LOCKSTATION		
PVSCOMMAND	LOCKSTATION	Message	Client mit Nachricht sperren
PVSCOMMAND	UNLOCKSTATION		
PVSLOGIN	USERNAME	username	Client Benutzername
PVSLOGIN	PASSWORD	password	Serverpasswort
PVSLOGIN	ID	id	
PVSLOGIN	FAILED	„Wrong Password“	Wird bei falschem Passwort gesendet
PVSCOMMAND	PORT	port	
PVSCOMMAND	PASSWORD	password	VNC Passwort
PVSCOMMAND	RWPASSWORD	rwpassword	Passwort für den Zugriff auf die Tastatur und Maus
PVSCOMMAND	VNC	YES	Erlaube Zugriff
PVSCOMMAND	VNC	NO	Verbiete Zugriff
PVSCOMMAND	PING		
PVSCOMMAND	VNCREQUEST		
PVSCOMMAND	VNCSRVRESULT	result code	Der Rückgabewert des pvs-vncsrv Skripts
PVSMESSAGE	BROADCAST	MESSAGE	
PVSMESSAGE	clientToAdd		Client hinzufügen (Chat)
PVSMESSAGE	clientToRemove		Client entfernen (Chat)
PVSMESSAGE	assignedName		Festgelegter Name (Chat)

Tabelle 8.1: Liste der PVS Messages

9 PVS-Steuerkonsole

Die Steuerkonsole ist das Hauptquartier des **Pool Video Switch**. Sie wird auf der Dozentmaschine oder einer geeigneten weiteren Maschine ausgeführt und sie ist für die Steuerung der Clients-Maschine zuständig. Alleine diese Rolle der Steuerkonsole setzt schon voraus, dass das GUI so benutzerfreundlich aufgebaut werden soll, dass diese Aufgabe vereinfacht wird.

Wir werden uns in den nächsten Punkten auf den technischen Aufbau der Steuerkonsole fokussieren. Wir werden als erstes die Auswahl von Qt zum Aufbau des Server-GUI, dann werden wir die GUI Server-Konsole vorstellen und zum Schluss werden wir die Beziehung Server-Client unter die Lupe nehmen.

9.1 pvsmgr in Qt

Das Server-GUI wurde erstmal in GTK/Gnome implementiert. Dank der Flexibilität, die von Qt ermöglicht wird, wurde die komplette Anwendung in Qt umgeschrieben. Qt ist ein UI Framework und ermöglicht die Implementierung von Anwendungen, die unter Linux (Unix), Windows, mobile und eingebettete Systemen ausführbar sind. Wir werden also den pvs-Manager(Server-GUI) komplett in Qt umschreiben und mit zusätzlichen Features gestalten.

9.2 GUI Server-Konsole

Hier sollte die Hauptsteuerung des PVS abgewickelt werden.

9.2.1 Die Architektur

Es wird u.a. möglich sein, Funktionen auszuführen wie: die Frame der einzelnen Clients ansehen, ein Profil anlegen, Screenshot aufnehmen, ein (Un)Projektion ausführen, eine Nachricht an Clients verschicken, usw... Die Grundstruktur des pvsmgr wird in der Abbildung 9.1 illustriert. Das Bild zeigt uns das Hauptfenster (*MainWindow*) der Anwendung und seine unterschiedlichen Komponenten:

MainWindow

Abbildung 9.1: pvsmgr-Architektur

Die Klasse MainWindow wird vom QMainWindow abgeleitet; das ist der Parent an der Spitze der Hierachie; Sie zu zerstören, bedeutet ein automatisches Schließen der einzelnen Komponente, also der gesammten Anwendung. Alle anderen Komponenten bauen auf ihr auf. Die Abbildung 9.2 zeigt ihr Abhängigkeitsdiagramm.
Alle andere Komponenten (Klasse) der Anwendung haben die Klasse MainWindow als Parent. Qt verwaltet eine Struktur der Art *parent-child* Beziehung, so dass wenn ein Parent zerstört wird, werden alle seine Kinder (*child*) automatisch mit zersört; Das ist der sogenannte *Domino-Effekt*.

Menubar

Diese Klasse enthält eine Liste von *pull-down*-Menu und QMainWindow speichert diese im QMenuBar. Die Klasse wird also vom QMenuBar abgeleitet. Die gespeicherten Menüs enthalten jeweils die Menu-Items, die nichts anderes als *QAction* sind. Beim Betätigen einer Aktion (Menü-Item) wird ein korrespondiertes *SIGNAL* gesendet, wir verbinden dann dieses zu einem *SLOTS*, wie hier gezeigt wird *connect(actionCreate-profile, SIGNAL(triggered()), MainWindow, SLOT(createProfile()))*; Also beim Clicken auf das Menü-Item "File->Profile manager"wird das Signal *triggered()* gesendet, das automatisch das Slot *createProfile()* aufruft. Genau dieses Prinzip wird auch auf die Aktionen im Toolbar angewandt.

Toolbar

Die Klasse wird vom QToolBatr abgeleitet und enthält eine Liste von Aktionen, die man unmittelbar auf die gesamten oder die ausgewählten Clients in der ConnectionList anwenden

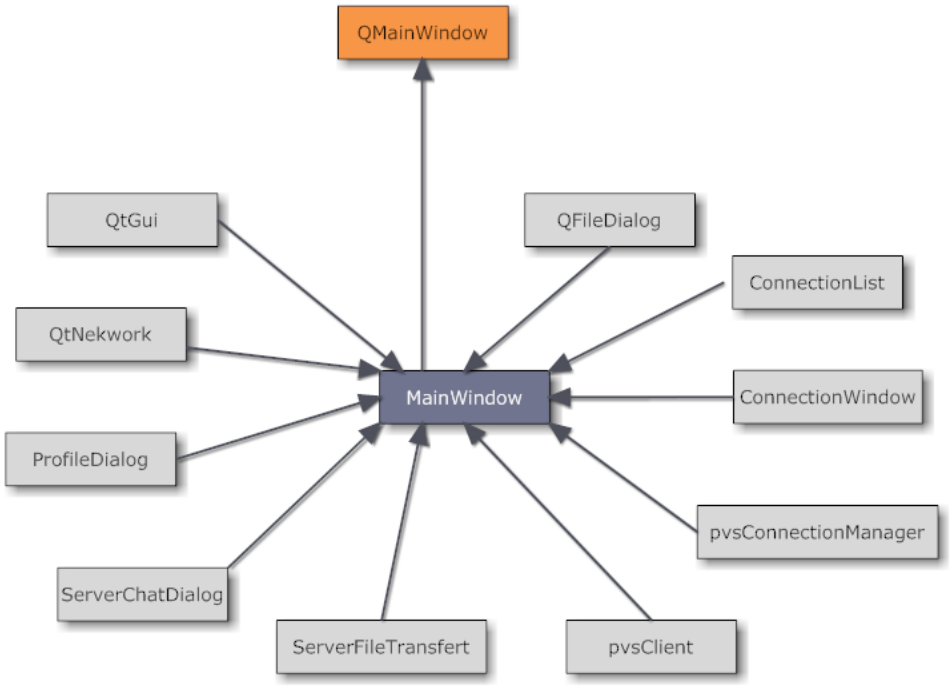

Abbildung 9.2: Abhängigkeitsdiagram: MainWindow

kann.

ConnectionList

Sie enthält die Liste der verbundenen Clients. Sie ist vom QTableView abgeleitet. Die vorhandenen Clients in der Liste können in unterschiedlichen Bezeichnungen angezeigt werden. Es ist also möglich die Clients unter verschiedenen Sichten anzuzeigen. Die Clients können mit Name, IP-Adresse oder Username angezeigt werden. Wir entnehmen also, dass das QTableView genauer gesehen 3 Spalten enthält, von denen nur eins davon immer sichtbar wird. Dieses Geschehen lässt sich ganz trivial ausführen. Man soll lediglich dafür sorgen, dass wenn eine Spalte angezeigt wird, sollen die 2 anderen ausgeblendet werden. Das QTableView steht zu diesem Zweck die Methode *setColumnHidden(int column, bool hidden)* zur Verfügung, wobei *column* die betroffene Spaltenummer (die erste Spalte hat den Index 0) ist und *hidden* legt dann fest, ob die Spalte ein-(*true*) oder ausgeblendet(*false*) bleiben soll. Angesichts des angewandten Selection-Model (*QItemSelectionModel*) ist es möglich mehrere Clients (*QAbstractItemView::ExtendedSelection*) aufzufassen und die entsprechenden verfügbaren Aktionen aus dem Tollbar oder der ContexMenu auszuführen.

ConnectionWindow

Die Klasse ist wie der zentrale Widget der Anwendung. Sie ist vom QWidget abgeleitet und enthält die Frame der jeweiligen verbundenen Clients. Klarerweise hat diese Komponente die MainWindow-Klasse als Parent.

ConsoleLog

Abgeleitet von QTextEdit, hier werden alle Logs angezeigt. Der Log-Bereich ist per Default ausgeblendet. Man kann ihn aber anzeigen lassen. Man unterscheidet mehrere Arte von Logs: Chat, Error, Normal, Network und Terminal.

Die richtig konfigurierten Clients werden mit dem pvs-Server verbunden und werden auf der GUI Server-Konsole angezeigt. Der Server bietet 2 unterschiedliche Sichten zur Anzeige der verbundenen Clients an.

Clientliste-Ansicht

Diese Sicht ist eine Liste, die aus einem *QTableview*-Object abgeleitet wird. Die Client-Liste ist definiert durch die ConnectionList-Klasse die Abbildung ?? zeigt uns die Abhängigkeits-diagramm dieser Klasse.

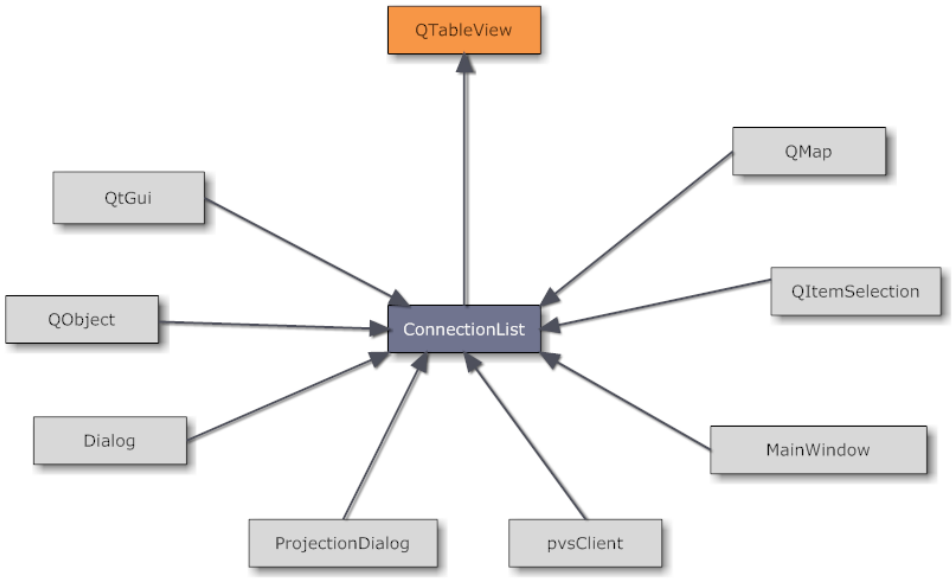

Abbildung 9.3: Abhängigkeitsdiagramm der ConnectionList-Klasse

Die Tabelle enthält 3 Spalten, von denen nur eine angezeigt werden kann. Mit Hilfe passender Tastenkombinationen können die anderen Spalten angezeigt werden. Die Spalten enthalten jeweils den Name der Clients, seine IP-Adresse und seinen Username. Außerdem haben wir hier ein Kontextmenü per Überschreibung der Methode *mouseReleaseEvent(QMouseEvent * e)* aus der Mutter-Klasse (QTableView) zur Verfügung gestellt. Das Kontextmenü ist hier im Grunde ein Popup-Menü, das Menü-Item enthält. Diese Menü-Items sind nichts anderes als Actionen und sie werden behandelt wie wir es schon oben beschrieben haben. Das Popup-Menü wird angezeigt, wenn die rechte Mause-Taste auf einem Item in *QTableView* losgelassen wird.
Von der jeweiligen Clients in der ConnectionList wollen wir jetzt die entsprechenden VNC-Ansichten vorstellen.

VNC-Ansicht

Die zweite Anzeigemöglichkeit, die vom Server angeboten wird ist die Anzeige der VNC-Verbindung zwischen Client und Server. Genauer gesagt, auf dem pvsmgr kann der Dozent jeder Zeit nachvollziehen, was auf der Client-Maschine passiert, dies wird vom rfb (Remote Frame Buffer) ermöglicht. Zu dem jeweiligen Client wird vom Manager ein *VNCClientThread()* initialisiert und in dieser Klasse wird der rfb-Client instanziiert. Die Klasse*VNCClientThread()* wird vom QThread abgeleitet, somit wird eine kontinuierliche Abfrage des Client rfb gesichert. Als QThread kann man mit der Methode *msleep(int updatefreq)* den Thread für eine bestimmte Zeit (in Millisekunde) schlafen lassen; dies ermöglicht also den pvsmgr die Abfragefrequenz des Client-rfb zu steuern. Darüberhinaus wird dem Dozent die Möglichkeit gegeben die Qualität der VNC-Qualität jeder Zeit festzulegen. Da es quasi nicht möglich ist, eine VNC-Qualität bei laufender Verbindung zu ändern, ist es dann notwendig dem Thread der entsprechenden Client zu terminieren und einen neuen mit dem gewünschten Quality zu starten und danach dem Client diesen neuen Thread wieder zuzuordnen.
Die VNC-Ströme werden in der ConnectionFrame-Klasse verwaltet. Die ConnectionFrame ist

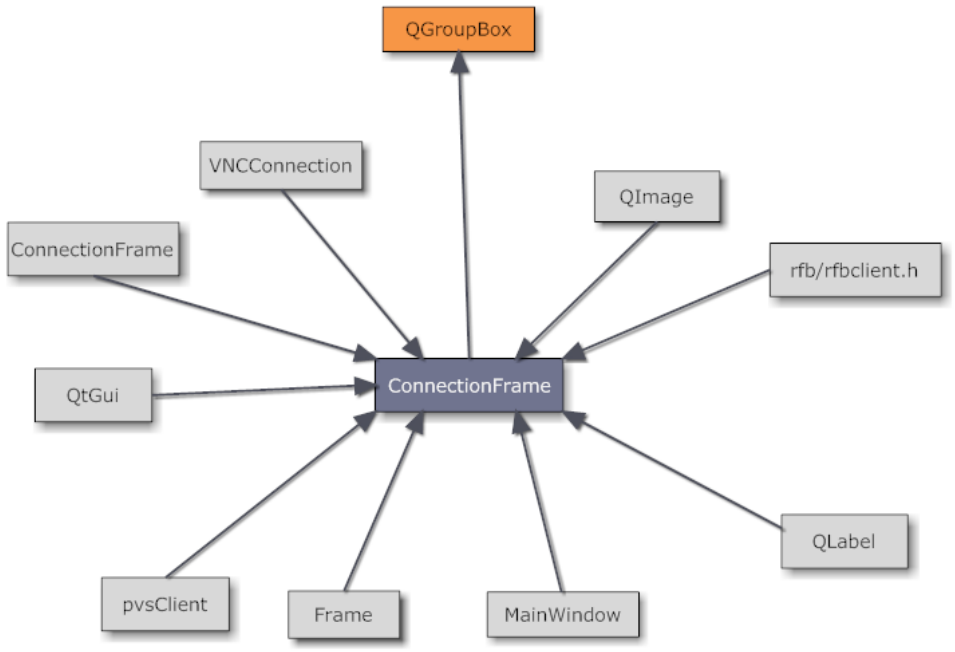

Abbildung 9.4: Abhängigkeitsdiagramm der ConnectionFrame-Klasse

vom QGroupBox abgeleitet. Die Abbildung ?? das Abhängigkeitsdiagramm der Klasse. Die Klasse wird vom QGroupBox abgeleitet und enthält das Object QLabel, das die eigentlichen VNC-Ströme anzeigt, denn QLabel ist im Qt eine optimale Unterstützung zum Anzeigen und zum skalieren von Bildern. Jede ConnectionFrame hat ein Kontextmenü und diese werden genau so wie wir es in der ConnectionList beschrieben haben, verwaltet.

Wir fahren mit der Vorstellung der pvsmgr-Struktur fort. Wir wollen uns in den nächsten Punkten der Verbindung zwischen Client und Server annähen, in dem wir dieser Aushandlung diese Verbindung betrachten. Allerdings werden wir nähere Details darüber in anderen

Abschnitten erläutern.

9.2.2 Client-Seite

Wir wollen davon ausgehen, dass im Subnetz, indem sich die Client-Maschine befindet, bereits ein pvsmgr gestartet wurde. Die Abwicklung auf dem server werden wir im nächsten Punkt erläutern. Nach dem Start eines Server sendet dieser eine Broadcast-Nachricht, die von allen verfügbaren Client erhalten wird. Erhält ein Client diese Nachricht, dann weiß er, dass ein Server gerade am laufen ist. Anschließend fügt der Client diesen Server in seiner Server-Liste hinzu. Wird es eine von einen anderen Server gesendetete Broadcast-Nachricht empfangen, wird ebenfalls den entsprechenden Server in die Server-Liste des Client hinzugenommen. Der Client verfügt also über eine Liste von potentiellen Server, mit denen er eine PVS-Verbindung aufbauen kann, die dann schließlich eine VNC-Verbindung wird und erst dann kann die Remote Frame Buffer initialisiert werden. Verläuft diese Initiliasierung erfolgreich, wird bei dem Server der VNC-Frame angezeigt. Aus seiner Server-Liste kann der Client freiwillig den Server auswählen, mit dem er sich verbinden möchte. Allerdings erfolgt die eigentliche Verbindung zwischen Server und Client erst, nachdem der Client das vom Dozent gesetzte Passwort richtig eingegeben hat. Beim Start des Server wird vom System ein zufälliges Passwort generiert und der Dozent soll dieses den Client im Raum bekanntgegeben, so dass ein anderer Client, der im Nebenraum steht und auch die Server-Broadcast-Nachricht erhalten hat, keine M0glichkeit hat, mit diesem Server eine Verbindung aufbauen zu können. Also nun hat ein Client das richtige Passwort eingeben und die Verbindung mit dem Server konnte erfolgreich aufgebaut werden. An dieser Stelle wollen wir noch betonen, dass der Dozent selbst entscheiden kann, ob ein Passwort zum Verbindungsaufbau mit dem PVS-Server vom Client aus eingegeben werden soll oder nicht. Dafür wird eine Checkbox auf dem pvsmgr-GUI eingebaut, die beim Ankreuzen per SIGNAL-SLOT die Funktion *PVSConnectionManager::getManager()->setNeedPassword(bool needed)* aufruft, wobei mit *needed=TRUE* wird ein Passwort benötigt.
Nun wollen wir erläutern wie der Server mit einer bestehenden Verbindung umgeht.

9.2.3 Server-Seite

Wir hatten schon erwähnt, dass der Server 2 Ansichtsmöglichkeiten zur Annzeige eines verbundenen Client verfügt. Nachdem ein Client mit dem Server eine Verbindung erfolgreich aufgebaut hat, wird fast zeitgleich erstmal der Client links in der Connection-Liste angezeigt und dann wird eine VNC-Verbindung zwischen Client und Server so aufgebaut, dass auf der rechten Seite (*Connectionwindow*) das Client-Frame angezeigt werden kann. Jeder Frame wird so beschriftet, dass es jeder Zeit möglich wird, aus einem Client in der Liste (links) den entsprechenden Frame (recht) anzuordnen.

Somit hat der Server einen besseren Überblick über die gesamten vorhandenen Verbindungen. Der Server kann also jegliche verfügbare Aktion ausführen. Der Betreiber (der Dozent) des pvsmgr verfügt über eine zweite Maschine, die an einen Beamer angeschlossen wird und auf dieser Maschine läuft ein PVS-Client. Dieser Client wird als *Superclient* bezeichnet und

verfügt dementsprechend gegenüber anderen Clients über einen Sonderstatus. Z.B: bei einer LockAll-Aktion wird er nicht gesperrt und dabei muss geachtet werden, dass ein auf dem Superclient projektierter Client auch nicht gesperrt wird.

9.3 Verbindungsverwaltung und Projektion

Meldet sich ein Client an der PVS-Steuerkonsole an, wird der Bildschirminhalt des Clients zur Thumbnail Darstellung angefordert. Hierzu sendet der Client seine Verbindungsdaten, d.h. Passwort, RW Passwort und Port, an die Steuerkonsole (s.h. Liste der PVS Messages 8.3.2). Die Verbindungsdaten werden in der Klasse `pvsConnectionManager` in einer Liste `std::list<PVSClient*> _listClients` von PVSClient Objekten verwaltet. Ein solches Clientobjekt enthält unter anderem Port, Passwort, RW Passwort sowie Flags, ob eine VNC Verbindung erlaubt ist, ob ein VNC Passwort vorhanden ist, und ob der Client gerade projeziert wird (siehe hierzu auch `/core/pvsClient.h`). Der pvsConnectionManager wird in allen Klassen verwendet, welche auf die Verbindungsdaten der Clients zugreifen müssen, daher ist er als Singleton implementiert und auf die entsprechende Instanz kann über `PVSConnectionManager::getManager()` zugegriffen werden.

9.3.1 Projektion

Um zu gewährleisten, dass bei einer Projektion ein als Quelle ausgewählter Client nicht als Ziel einer Projektion gewählt werden kann, werden in der Klasse `ConnectionList` eine Liste `QList<QString> targetList` mit den Projektionszielen und eine Map `QMap<QString, QList<QString> > sourceMap` mit der Quelle und der zugehörigen Liste von Zielen geführt. Wird ein Client als Projektionsquelle ausgewählt, wird zunächst überprüft, ob dieser Client schon in der Liste der Projektionsziele vorhanden ist. Ist er nicht vorhanden, wird ein Auswahldialog `projectionDialog` mit allen verfügbaren Projektionszielen angezeigt (ProjectionDialog erhält die entsprechende Liste durch `MainWindow::getWindow()->getConnectionList()->getTargetToDisplay(source)`). Projektionsziele können hierbei alle angemeldeten Clients sein, die nicht Ziel oder Quelle einer anderen Projektion sind. Die Auswahl der Ziele im `projectionDialog` erfolgt durch Checkboxen. Die hierdurch gewählten Clients werden zurück an die den `projectionDialog` aufrufende Klasse übergeben. Nachdem Quelle und Ziel ausgewählt wurden, wird die Funktion `projectStations`, welche zunächst prüft, ob auf dem Quell-Client überhaupt ein VNC Server läuft, in der Klasse `ConnectionWindow` mit dem Quell-Client als Parameter aufgerufen. Ist ein VNC Server gestartet, so werden die Verbindungsdaten des Quell-Clients, die über die get-Methoden des entsprechenden PVSClient Objekts ausgelesen werden, an alle Ziel-Clients als durch Leerzeichen separierte Liste gesendet. Auf Clientseite werden diese Verbindungsdaten dann in ein Array transformiert und an den `clientVNCViewer` übergeben (via DBus). Soll ein Client als Ziel zu einer existierenden Projektion hinzugefügt werden, wird der entsprechende Quellclient und alle neuen bzw. zusätzlichen Projektionsziele gewählt. Die Zielclients werden dann einfach der in der `sourceMap` gespeicherten Liste der Quelle sowie der `targetList` hinzugefügt.

Qualitätsoptionen

Zusätzlich zu den Verbindungsdaten werden Qualitätsoptionen zu der gesendeten Nachricht hinzugefügt. Die Qualitätsoption ist ein Integerwert 0,1 oder 2 wobei 0 der besten Qualität, 2 der schlechtesten Qualität entspricht. Dieser Wert wird an den Zielclient einer Projektion gesendet, welcher die Qualitätseinstellungen des VNC-Viewers entsprechend anpasst. Aktuell wird die Qualität immer auf 0 gesetzt, da noch die Einstellungsmöglichkeiten in der GUI fehlen.

9.3.2 Remote Help

Remote Help funktioniert analog zur Projektion, jedoch ist hier zu gewährleisten, dass jeweils nur ein Quellclient und ein Zielclient ausgewählt werden, da es nicht sinnvoll ist, Tastatur und Maus eines Clients von mehreren anderen Clients aus zu steuern. Anstelle des Passworts wird das RW-Passwort des Clients verwendet. Die Möglichkeit, die Maus und die Tastatur eines Clients zu steuern, ist mit dem verwendeten VNC-Server schon gegeben, für die Steuerkonsole ändert sich hier nichts, lediglich der VNC Viewer muss die Tastatureingaben und die Mausbewegung an den VNC Server weitergeben (siehe hierzu auch 10.7.1 Tastatur und Maussteuerung).

10 PVS-Client

Der PVS-Client ist in ein Backend (**pvs**) und ein Frontend (**pvsgui**) aufgeteilt. Erster läuft als Daemon im Hintergrund und kommuniziert über ein gegebenes Netzwerk mit der Steuerkonsole (**pvsmgr**). Die Interprozesskommunikation (inter-process communication, IPC) zwischen Back- und Frontend erfolgt über D-Bus und wurde mit Hilfe der von Qt4 bereitgestellten Bibliothek QtDBus realisiert. Um das Backend von einer GUI komplett unabhängig zu halten, wurde darauf geachtet, dass alle Nachrichten, die an die GUI gerichtet sind, vom Backend als Signale über D-Bus verteilt werden. Umgekehrt, wenn das Frontend eine Nachricht an das Backend schicken will, muss es sich der im Backend implementierten Slots bedienen (s. Abbildung 10.1). Somit ist es sogar möglich, dass mehrere GUIs gleichzeitig ein und das selbe Backend benutzen. Ein weiteres Feature ist, dass das Frontend beim Start das Backend durch einen D-Bus Aufruf automatisch startet. Hierzu muss allerdings die Applikation zuvor mit **make install** auf dem System installiert worden sein.

Zunächst muss in der Hauptklasse des Backends (*pvs.cpp*) das Qt-Makro

```
Q_CLASSINFO("D-Bus Interface","org.openslx.pvs")
```

eingefügt werden, um dem späteren D-Bus-Interface einen Namen zu geben. Als nächstes werden alle zu exportierenden Slots wie gewohnt mit `Q_SLOTS` bzw. Signale mit `Q_SIGNALS` in der *pvs.h* gekennzeichnet. Schließlich folgen ein paar Definitionen in der *CMakeLists.txt*, deren Bedeutung aus den Kommentaren entnommen werden kann:

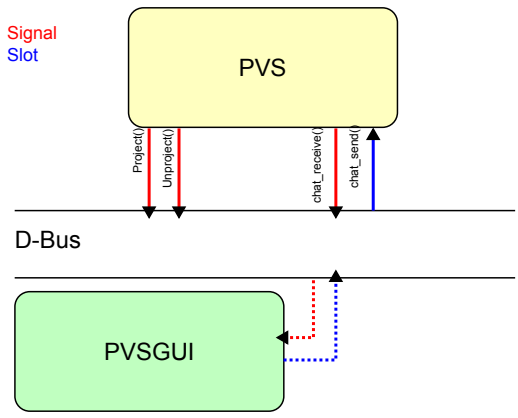

Abbildung 10.1: Back- und Frontend des PVS-Clients kommunizieren über D-Bus

```
# QtDBus Modul aktivieren
SET( QT_USE_QTDBUS TRUE )
# Signale und Slots aus pvs.h ins XML-Format exportieren
QT4_GENERATE_DBUS_INTERFACE( src/pvs.h org.openslx.pvs.xml )
# D-Bus Adapter für Backend erzeugen
QT4_ADD_DBUS_ADAPTOR( PVS_SRCS ${CMAKE_BINARY_DIR}/org.openslx.pvs.xml
  src/pvs.h PVS )
# Aus der zuvor erstellten XML-Datei ein Interface in C++ Syntax
# generieren
QT4_ADD_DBUS_INTERFACE( PVSGUI_SRCS
  ${CMAKE_BINARY_DIR}/org.openslx.pvs.xml pvsinterface )
```

Sind die Vorbedingungen erfüllt, wird beim ersten Kompilieren eine Datei namens *pvsinterface.h* erstellt, in der die Definition der Klasse `OrgOpenslxPvsInterface` gespeichert wurde. Diese Klasse kann nun in anderen Programmen (wie z.B. in der **pvsgui**) dazu benutzt werden, um Zugriff auf die freigegebenen Ressourcen des Backends zu erhalten.

10.1 Grafische Benutzeroberfläche

Die Hauptklasse der Benutzeroberfläche des Clients heißt `PVSGUI` und befindet sich in der Datei *pvsgui.cpp*. In dieser werden Referenzen auf alle benutzten Dialoge, das System Tray Icon sowie das D-Bus Interface zur Kommunikation mit dem Backend erzeugt. Weiter beinhaltet diese Klasse die `main()`-Funktion und bildet somit den Einstiegt für das Betriebssystem. Damit der Zugriff auf die Toolbar immer gewährleistet bleibt (auch wenn Videos oder eine virtuelle Maschine im Vollbild laufen), muss beim Zeichnen dieser der laufende Windowmanager umgangen werden. Hierbei hilft uns Qt mit der Anweisung:

```
setWindowFlags(Qt::WindowStaysOnTopHint|Qt::X11BypassWindowManagerHint);
```

Wie bereits in der Einleitung erwähnt, wird das komplette Projekt mit Hilfe von CMake erstellt. Aus diesem Grund soll an dieser Stelle ebenfalls eine kleine Beschreibung folgen, wie man mit CMake Qt-Klassen einbindet, die mit dem Qt-Werkzeug **designer** erstellt worden sind. Als Beispiel dient die Klasse `PVSGUI`, wobei wir hier nicht auf die Qt eigenen Details eingehen werden.

```
# CMake anweisen, Qt4 auf lokalem System zu suchen
FIND_PACKAGE( Qt4 4.5.0 REQUIRED )
INCLUDE( ${QT_USE_FILE} )
# Sowohl Quell- als auch Zielverzeichnis auf Quelldateien durchsuchen
INCLUDE_DIRECTORIES( ${CMAKE_SOURCE_DIR} ${CMAKE_BINARY_DIR} )

# Alle von der Software benötigten Quelldateien
SET( PVSGUI_SRCS src/pvsgui.cpp )
# Vom Meta Object Compiler benötigte Header-Dateien
SET( PVSGUI_MOC_HDRS src/pvsgui.h )
# Auflistung aller benötigten .ui Dateien
```

```
SET( PVSGUI_UIS src/gui/ui/clientToolbar.ui )
# Ressourcendateien
SET( PVSGUI_RCS pvsgui.qrc )

# Qt Meta Object Compiler ausführen (moc)
QT4_WRAP_CPP( PVSGUI_MOC_SRCS ${PVSGUI_MOC_HDRS} )
# Qt User Interface Compiler ausführen (uic)
QT4_WRAP_UI( PVSGUI_UI_HDRS ${PVSGUI_UIS} )
# Qt Resource Compiler ausführen (rrc)
QT4_ADD_RESOURCES( PVSGUI_RC_SRCS ${PVSGUI_RCS} )

# Applikation mit allen benötigten Variablen bauen
ADD_EXECUTABLE( pvs ${PVS_SRCS} ${PVS_MOC_SRCS} ${PVS_RC_SRCS} )
# Mit der Qt-Bibliothek linken
TARGET_LINK_LIBRARIES( pvsgui ${QT_LIBRARIES} )
```

10.2 User-Interface für Benutzerkonfiguration

Der eingebaute Konfigurationsdialog besteht hauptsächlich aus der Klasse `ClientConfigDialog` und ist in der Datei *clientConfigDialog.cpp* definiert. Es handelt sich hierbei um einen gewöhnlichen `QDialog`, dessen graphische Implementierung in der Datei *clientConfigDialog.ui* gespeichert ist.
Den Kern dieser Klasse bildet ein `QSettings` Objekt, das sowohl zum Speichern als auch zum Laden der Benutzerkonfiguration benutzt wird. Ablage aller Einstellungen ist die Datei *~/.config/openslx/pvsgui.conf*. Diese kann jedoch zentral für die gesamte Applikation in der Funktion `PVSGUI::main(int argc, char *argv[])` geändert werden.
Um weitere Komponenten der GUI über Änderungen der aktuellen Konfiguration zu informieren, wird das Signal `configChanged()` emittiert.

10.3 Darstellung von VNC-Datenströmen

Die übertragenen Bildinformationen können in einem separaten Fenster oder im Vollbildmodus dargestellt werden. Hierzu wählt der Dozent über die Steuerkonsole (**pvsmgr**) eine Quelle und ein oder mehrere Ziele zur Projektion aus, was zur Folge hat, dass an alle gewählten Clients (**pvs**) eine Nachricht mit den benötigten Informationen geschickt wird. Als nächstes sendet das Backend ein Signal über D-Bus an sein Frontend, das die benötigten Informationen (Host, Port, Passwort, Qualität) enthält. Somit kann nun die GUI eine Verbindung zum gegebenen VNC-Server aufbauen und den Datenstrom darstellen.

Der Aufbau des VNC-Viewers besteht aus zwei Klassen:

- Die Klasse `VNCClientThread` stellt mit Hilfe eines `rfbclient` (aus libvncserver0) eine Verbindung zum VNC-Server her. Zunächst werden sämtliche Verbindungsparameter über den Konstruktor definiert und nach einem Aufruf von `VNCClientThread::start()`

eine Verbindung aufgebaut. Da diese Klasse selbst von `QThread` abgeleitet ist, läuft sie in einem eigenständigem Thread, um die GUI nicht zu blockieren. Der Informationsaustausch erfolgt über das Signal `VNCClientThread::imageUpdated(int x, int y, int w, int h)`, welches die Koordinaten einer Änderung des Framebuffers enthält.

- Die Darstellung des Datenstroms wird von der Klasse `ClientVNCViewer` übernommen. Da beide Klassen auf ein und den selben Ressourcen arbeiten, müssen die Signale des `VNCClientThread` mit dem Parameter `Qt::BlockingQueuedConnection` verbunden werden. Dies stellt sicher, dass der Thread solange pausiert, bis der Slot des `ClientVNCViewer` abgearbeitet wurde (`ClientVNCViewer::updateImage(int x, int y, int w, int h)`).

Bei der Entwicklung des VNC-Viewers musste stark auf die Performanz geachtet werden. Das Ziel ist es, sowohl Bandbreite als auch CPU-Last einzusparen. Um die Netzwerkauslastung konfigurierbar zu machen, stehen drei Qualitätsstufen zur Wahl, von denen eine gewählt werden muss. Durch diese serverseitige Konfiguration wird beispielsweise auf der niedrigsten Qualitätsstufe die Farbtiefe verringert und die Kompressionsstufe erhöht (verlustbehaftet), was die zu übertragenen Bildinformationen minimiert. Um den Prozessor des Zielrechners nicht unnötig zu belasten, werden wie im VNC-Protokoll vorgesehen, nur die Bereiche des Framebuffers neu gezeichnet und skaliert, die sich auch geändert haben.

10.4 Chat-Interface

Die vom Chat-Interface verwendeten Klassen und Dialoge befinden sich in den Dateien *clientChatDialog.cpp* und *clientChatDialog.ui*. Der Informationsaustausch erfolgt hier wieder über D-Bus direkt mit dem Backend.
Zum Versenden einer Nachricht wird der Slot `PVS::chat_send()` und zum Empfang das Signal `PVS::chat_receive()` des Backends benutzt. Um die Liste aller Chatteilnehmer aktuell zu halten, wird analog vorgegangen. Hierzu existieren die Slots `PVS::chat_client_add()` und `PVS::chat_client_remove()`.
Um neben dem öffentlichen Chat auch private Unterhaltungen zwischen zwei Teilnehmern zu ermöglichen, wurde ein `QTabWidget` benutzt. Auf diesem wird bei jeder neu begonnenen Unterhaltung ein `QTextEdit` platziert, dass dem Gesprächspartner durch ein `QHash<QString, QTextEdit*>` direkt zugeordnet wird.

10.5 Dateiübertragung und Interface

Die graphische Benutzerschnittstelle des Clients ermöglicht allen Chatteilnehmern binäre Dateien untereinander zu tauschen. Dazu wurden zwei Dialoge entwickelt – einer zum Senden und einer zum Empfang. Diese Dialoge stellen nicht nur das Benutzerinterface dar sondern bilden ebenfalls die komplette Grundlage (Client-Server) zur Übertragung binärer Daten. Die hierfür entworfenen Klassen sind `ClientFileSendDialog` und `ClientFileReceiveDialog`, zu denen die gleichnamigen .ui Dateien gehören.

Daten senden:
Um eine Datei zu senden, sind der Name des Chatteilnehmers (Nickname) sowie der Pfad der zu sendenden Datei erforderlich. Falls diese Informationen nicht vorhanden sind, wird der Benutzer durch Pop-Ups danach gefragt. Der nächste Schritt besteht darin, die IP bzw. den Hostnamen des Chatteilnehmers zu erfragen. Dies geschieht automatisch, indem das Backend (**pvs**) diesbezüglich über D-Bus befragt wird. Stehen alle Angaben zur Verfügung, wird an den Zielrechner ein kleiner Header mit folgenden Informationen geschickt:

```
Nickname_des_Senders;Dateiname;Dateigröße\n
```

Sollte sich der Kommunikationspartner für den Empfang der angebotenen Datei entscheiden, wird von diesem eine Bestätigung verschickt (ack), die den eigentlichen Übertragungsvorgang startet. Das Ende der Übertragung wird dem Empfänger durch das Schließen des benutzen Sockets signalisiert.
Während der Übertragung wird ein Dialog angezeigt der den Sender über den Fortschritt mit Hilfe einer `QProgressBar` informiert. Der Vorgang kann jederzeit abgebrochen werden und über aufgetretene Fehler wird durch Pop-Ups informiert.

Daten empfangen:
Sobald die graphische Benutzeroberfläche des Clients startet, wird durch eine Instanz des `QTcpServer` der Port 29481 geöffnet und auf eingehende Verbindungen gewartet. Wird nun von einem anderen Teilnehmer eine Datei angeboten, so wird eine Instanz des `ClientFileReceiveDialog` angelegt, die sich um den weiteren Verlauf kümmert. Der Empfänger kann nun, die Übertragung akzeptieren und einen Speicherort wählen. Danach wird eine Bestätigung an den Sender verschickt und die eigentliche Datenübertragung gestartet.
Auch der Empfänger wird durch einen Dialog über den Fortschritt informiert und kann die Übertragung jederzeit abbrechen.

Wichtig an dieser Stelle ist die Speicherfreigabe beendeter Übertragungsdialoge. Da wir gleichzeitig beliebig viele Übertragungen erlauben wollen, muss natürlich für jede ein eigenständiger Dialog erzeugt werden. Um keine Liste mit laufenden Dialogen führen zu müssen (und einzelne zu löschen) wird hier auf einen komfortablen Qt-Mechanismus zurückgegriffen. Sobald ein Dialog mit `accept()` oder `reject()` beendet wird, sendet dieser das Signal `finished(int)`. Auf der anderen Seite besitzt jede Klasse, die von `QObject` abgeleitet wurde, den Slot `deleteLater()`. Die Qt-API garantiert, dass Qt nach dem Aufruf dieser Methode dieses Objekt selbstständig aus dem Speicher entfernt. Also müssen wir dieses Signal dem Slot zuordnen (in `ClientFileSendDialog` und `ClientFileReceiveDialog`):

```
connect(this, SIGNAL(finished(int)), this, SLOT(deleteLater()));
```

10.6 VNC Server

Um den Bildschirminhalt von Clients zum einen in der Übersicht (Thumbnails) der Steuerkonsole und zum anderen auf anderen Clients und dem Beamer darstellen zu können, wird das sogenannte Remote Framebuffer Protokoll (RFB), welches auch der Virtual Network Computing (VNC) Software zugrundeliegt, benutzt. Um VNC bzw RFB nutzen zu können, wird

ein VNC Server und ein VNC Viewer benötigt. Aktuell wird als VNC Server `x11vnc` verwendet sowie der, wie in 10.3 beschrieben, in den den PVS-Client integrierte rfbclient bzw. ClientVNCViewer. Bei der VNC Verbindung wird zwischen einer Verbindung ohne Maus und Tastatursteuerung, dem sogenannten Viewonly oder Readonly Modus, und einer Verbindung mit Maus und Tastatursteuerung, im folgendenden Read-Write (RW) Modus genannt, unterschieden.

10.6.1 Vergleich von VNC Servern

Es gibt zwei Arten von VNC Servern. Server, die eine laufende Sitzung bzw. ein vorhandenes Display darstellen bzw. übermitteln können, und Server, die eine eigene Sitzung bzw. ein eigenes Display bereitstellen (virtuell). Für PVS ist die zweite Art, welche auf dem ursprünglichen VNC basiert, nicht verwendbar, da man hierbei nicht den aktuellen Bildschirminhalt der Clients in der Steuerkonsole anzeigen bzw. projezieren kann. Daher wurden folgende VNC Server im Weiteren nicht betrachtet: Xvnc (RealVNC), vnc4server, TightVNC.
Motivation für den eigentlichen Vergleich ist die Integration mancher VNC Server in die Desktopumgebung. Hier müsste der x11vnc nicht extra installiert werden. Wichtig für den Einsatz im PVS ist vor allem die Unterstützung von Shared Modus, d.h. mehrere VNC Viewer können sich gleichzeitig mit dem VNC Server verbinden, und Forever Modus, d.h. der VNC Server wird nicht nach Trennung des letzten Clients beendet. Sicherheitsoptionen wie die Vergabe von Passwörtern und die Unterscheidung von RW und Viewonly Modus sind ebenso nicht zu vernachlässigen.

Vino
Vino ist der VNC Server der Gnome Desktopumgebung. Er wird standardmäßig über die graphische Oberfläche **vino-preferences** konfiguriert. Über die Kommandozeile ist Vino entweder über **gconftool-2** oder über direktes editieren der Datei *~/.gconf/desktop/gnome/remote_access/%gconf.xml* konfigurierbar. Parameter können nicht übergeben werden, was die Verwendbarkeit im PVS einschränkt. Vino benutzt standardmäßig den Port 5900, um auf eingehende Verbindungen zu warten. Dieser Port kann nur durch `gconftool-2 -s -t int desktop/gnome/remote_access/alternative_port <portnumber>` bzw. das Einfügen von z.B. `<entry name=“alternative_port” mtime=“1259858032” type= “int” value=“<portnumber>”/>` in die entsprechende %gconf.xml Datei geändert werden. Hierbei ist zu beachten, dass nicht überprüft wird, ob der entsprechende Port schon belegt ist. Dementsprechend wird auch kein anderer freier Port gewählt. Außerdem wird eine Änderung des Ports erst nach dem Neustart der grafischen Oberfläche aktiv. Vino unterstützt die Vergabe eines Passworts, welches base64 kodiert in der gconf Datei abgelegt wird. Ebenso kann man den Zugriff auf den Viewonly Modus beschränken. Unterschiedliche Passwörter für den RW Modus und den Viewonly Modus können jedoch nicht vergeben werden. Der Shared und Forever Modus sind standardmäßig aktiv.

Krfb
Krfb ist das Pendant zu Vino der KDE Desktopumgebung und kann entweder über eine Kon-

figurationsdatei oder über die grafische Oberfläche konfiguriert werden. Es kann entweder das Einladungssystem verwendet werden, bei dem eine Einladung mit einem Einmalpasswort generiert wird, oder man erlaubt uneingeladene Verbindungen und vergibt ein eigenes Passwort. Um Krfb über eine Konfigurationsdatei mit einem selbstgewählten Passwort für uneingeladene Verbindungen zu konfigurieren, erstellt man die Datei wie im folgenden angegeben und übergibt sie an krfb mit `krfb -config <Pfad>` bzw überschreibt oder bearbeitet die eigentliche Konfigurationsdatei in */.kde/share/config/krfbrc*.

```
[Security]
allowDesktopControl=false
allowUninvitedConnections=true
uninvitedConnectionPassword=<passwort>

[TCP]
port=<port>
useDefaultPort=false
```

Mit allowDesktopControl kann zwischen RW und Viewonly Modus gewechselt werden, unterschiedliche Passwörter können hierfür nicht vergeben werden. Zu beachten ist, dass die Passwörter für uneingeladene Verbindungen unverschlüsselt gespeichert werden. Wie auch Vino läuft Krfb standardmäßig im Shared und Forever Modus.

x11vnc

x11vnc ist ein auf libvncserver basierender VNC-Server, der von Karl Runge entwickelt wurde, um unter Linux/Unix Systemen existierende Displays über VNC anzeigen zu lassen. x11vnc wird über Parameter von der Kommandozeile aus konfiguriert. Eine ausführliche Liste aller verfügbaren Parameter findet sich unter http://www.karlrunge.com/x11vnc/x11vnc_opts.html. Zu beachten ist hierbei, dass alle Parameter, die an x11vnc übergeben werden (also beispielsweise auch Passwörter), in der z.B. durch **ps aux** abrufbaren Prozessliste anzeigbar sind. Daher sollte eine Passwortdatei, welche auch nach dem Auslesen automatisch von x11vnc gelöscht werden kann (s.h. 10.6.2), benutzt werden. Die Struktur einer solchen Passwortdatei ist wie folgt:

```
 rwpassword1
        .
        .
 [rwpasswordX]
 [\_\_BEGIN_VIEWONLY\_\_]
 [password1]
  .
.
 [passwordX]
```

Es muss mindestens ein Passwort vorhanden sein. Alle Passwörter bis zu der Zeile, welche __BEGIN_VIEWONLY__ enthält, werden als rw-Passwörter angesehen (außer in dem Spezialfall, dass nur zwei Passwörter und keine Zeile mit BEGIN_VIEWONLY vorhanden

sind, dann wird das zweite Passwort automatisch als Viewonly Passwort behandelt). Alle Passwörter nach der entsprechenden Viewonly-Zeile werden als Viewonly Passwörter gesehen. So ist es möglich beliebig viele Viewonly und bzw. oder RW-Passwörter zu generieren.

10.6.2 VNC Script

Der VNC Server wird durch ein Bash-Skript (*/misc/pvs-vncsrv*) von der Klasse *pvs* gestartet. Es wird zufällig jeweils ein Passwort für Standardzugriff und ein Passwort für den Zugriff mit Maus und Tastaturunterstützung generiert. Die generierten Passwörter werden dem Skript als Parameter übergeben. **pvs-vncsrv** wird zum Starten wie folgt aufgerufen: `pvs-vncsrv start port passwort [rwpasswort]`. Der Parameter `rwpasswort` ist optional und kann beim Starten des x11vnc zur Verwendung im Viewonly Modus weggelassen werden. Um den VNC Server zu stoppen, kann das VNC Script mit dem Parameter `stop` aufgerufen werden. Die übergebenen Passwörter werden zunächst auf folgende Weise in die Datei */.pvs/vncpassword* geschrieben: `rwpasswort __BEGIN_VIEWONLY__ passwort`. Als nächstes wird x11vnc mit folgenden Parametern ausgeführt:

- -auth ... - die Xauthority Datei
- -bg - im Hintergrund starten
- -forever - den X Server nicht beenden, wenn sich ein Client trennt
- -display :0 - das anzuzeigende Display
- -passwdfile rm:... - Pfad der Passwortdatei
- -o ... - Pfad der Logdatei (/.pvs/log.vncsrv)
- -shared - erlaube mehreren Clients, sich zu verbinden

Zu beachten ist vor allem das `rm:` vor der Pfadangabe bei `-passwdfile`. Dies bewirkt das Löschen der Passwortdatei, nachdem x11vnc sie eingelesen hat. Die Passwortdatei wird verwendet, da man ansonsten Passwörter, die nur als Parameter im Klartext an x11vnc übergeben werden, mittels **ps aux** lesen könnte.
Um einen Mehrfachstart des VNC Servers zu verhindern, wird zunächst immer ein `pvs-vncsrv stop` ausgeführt, welches nach der Prozessid des x11vnc sucht und diesen mit kill -9 beendet.

10.7 VNC Viewer

Der VNC Viewer ist, wie in der 10.3 Darstellung von VNC-Datenströmen beschrieben, implementiert. Die direkte Integration des Viewers im Gegensatz zum Einsatz eines externen VNC Servers bietet sich hier wegen der Integrationsmöglichkeiten in die GUI an (der VNC-Server ist GUI unabhängig). Jedoch muss der VNC-Viewer zur Unterstützung des RW-Modus des VNC-Servers noch verändert werden.

10.7.1 Tastatur und Maussteuerung

Um die Maus- und Tastatureingaben an den VNC-Server weiterzuleiten, müssen zunächst die Eingaben in der Klasse `ClientVNCViewer` abgefangen und verarbeitet werden. Dies kann durch Überschreiben der geerbten Methode `event` erreicht werden `bool ClientVNCViewer::event(QEvent *event)`. Hier kann durch `event->type()` der Typ des Events herausgefunden werden. Zu behandelnde Eventtypen sind:

- QEvent:KeyPress - Eine Taste wurde gedrückt
- QEvent:KeyRelease - Eine Taste wurde losgelassen
- QEvent:MouseButtonDblClick - Mausdoppelklick
- QEvent:MouseButtonPress - Maustaste gedrückt
- QEvent:MouseButtonRelease - Maustaste losgelassen
- QEvent:MouseMove - Maus wurde bewegt
- QEvent:Wheel - Mausrad wurde bewegt

Hier können nun die jeweils zuständigen Funktionen aufgerufen werden. Dabei ist zu beachten, dass die Methode `event()` bei allen Eingaben, die von einer eigenen Funktion behandelt werden, true zurückgibt (d.h. das Event wurde akzeptiert und behandelt). In den aufgerufenen Methoden kann dann zwischen den verschiedenen Maustasten (Qt:LeftButton, Qt:MidButton, Qt:RightButton) und den verschiedenen Tasten der Tastatur (z.B. Qt:Key_A, Qt:Key_B, Qt:Key_Alt siehe hierzu auch http://doc.trolltech.com/4.6/qt.html#Key-enum) unterschieden werden. Die Tastatur- und Mauseingaben müssen dann entsprechend an `vncClientThread` weitergeleitet werden. In der Klasse vncClientThread werden die Eingaben dann in Objekte gekapselt in einer Queue, welche regelmäßig geleert wird, gespeichert. Die Objekte sind dabei zum einen ein `PointerEvent` und ein `KeyEvent`. `PointerEvents` enthalten die Koordinaten des Mauszeigers (x,y) und einen hexadezimalen Wert, der die Maustasten und deren Zustand repräsentiert. Werte sind dabei:

- 0x01 = linke Maustaste gedrückt
- 0x02 = mittlere Maustaste gedrückt
- 0x03 = rechte Maustaste gedrückt
- 0xfe = linke Maustaste losgelassen
- 0xfd = mittlere Maustaste losgelassen
- 0xfb = rechte Maustaste losgelassen

KeyEvents bestehen aus einem hexadezimalen Wert, der die Taste repräsentiert, und True oder False je nachdem, ob die Taste gedrückt wurde oder nicht. Die hexadezimalen Werte entsprechen den Werten in einem X Window System und können in der Headerdatei <X11/keysymdef.h> nachgeschlagen werden (siehe hierzu auch http://www.realvnc.com/docs/rfbproto.pdf - RFB Protokollspezifikation, Abschnitt KeyEvent). Beim Leeren der Queue werden die Methoden der rfbclient Klasse der libvnc Bibliothek SendPointerEvent(cl, _x, _y, _buttonMask) und SendKeyEvent(cl, _key, _pressed) mit den entsprechenden Werten aufgerufen (cl entspricht dabei einem Pointer auf die eigentliche rfbClient Instanz).

10.8 Signalbehandlung

Um zu gewährleisten, dass bei einer Terminierung des PVS-Clients auch der gegebenenfalls gestartete VNC-Server gestoppt wird, werden Sigterm, Sighup, Sigquit und Sigint Signale abgefangen und weiterbehandelt. Um die Signale behandeln zu können, muss die C Bibliothek signals.h eingebunden werden und ein sigaction Objekt erstellt werden. Das Feld sa_handler des sigaction Objektes gibt dabei die aufzurufende Funktion an (Pointer auf die Funktion). Diese Funktion (in der Klasse pvs die Funktion signalHandler) muss dabei vom Typ void sein und die Signalnummer als Parameter erwarten (void PVS::signalHandler(int signal)).

```
struct sigaction act;
act.sa_handler = &PVS::signalHandler;
```

Um Signale abfangen zu können, muss nun noch das sigaction Objekt durch sigaction(SIGTERM, &act, 0) mit den entsprechenden Signalen (hier SIGTERM) verknüpft werden (muss für alle zu behandelnde Signale durchgeführt werden). Danach wird, sobald ein entsprechendes Signal erhalten wird, die Funktion aufgerufen und die Signalnummer übergeben. Für alle Signale kann die gleiche Funktion aufgerufen werden, hier kann die Unterscheidung dann mittels der in signal.h vorhandenen Konstanten SIGHUP, SIGTERM, SIGQUIT usw. erfolgen. Zu beachten ist hierbei, dass ein SIGKILL nicht abgefangen werden kann.

Teil IV

Anhang

Klassendiagramm