Universität Hamburg

Projektbericht

StorageTek L180

Veranstaltung: Projekt Parallelrechnerevaluation
Veranstaltungsnummer: 64-194
Veranstaltungskategorie: Projekt
Betreuer: Dr. Julian Kunkel

Eingereicht von:

Name:	Marc Dennis Perzborn
Fachsemester:	3
Matrikel-Nr.:	6554028
Adresse:	Bramfelder Chaussee 62 22177 Hamburg
E-Mail:	marcperzborn@gmx.de
Telefon:	0174/5666324
Fachrichtung:	Chemotechnik
Unterrichtsfach:	Informatik

Hamburg, den 20.Novemeber 2015

Inhaltsverzeichnis

1 Einleitung

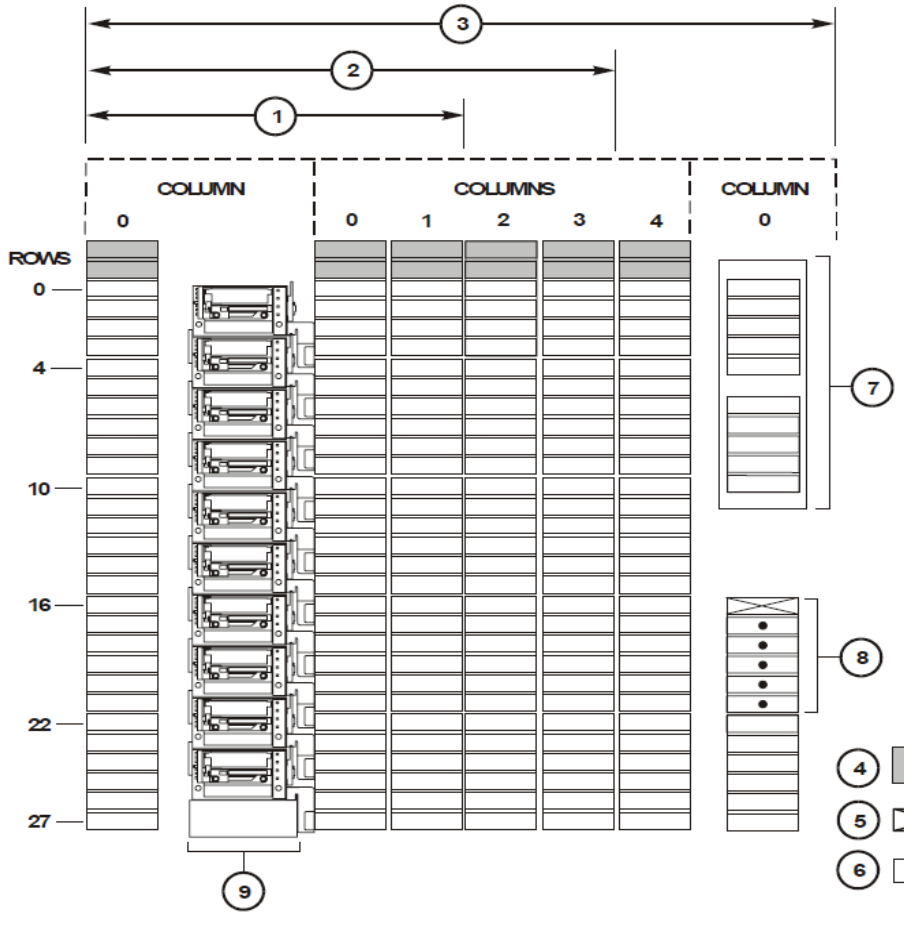
Ich werde über mein Projekt zur Tape Library StorageTek L180 berichten. Das Ziel dieses Projektes war es eine Kommunikation zwischen dem Rechner und der Tape Library herzustellen. Das Projekt wurde auf einem Rechner des Deutschen Klimarechenzentrums (DKRZ) mit dem Betriebssystem Ubuntu durchgeführt. Die physikalische Verbindung zwischen Rechner und Library wurde über Fibre Channel hergestellt. Es sollte ein Programm in Python geschrieben werden, welches eine grafische Visualisierung der Tape Library mit allen vorhandenen Slots beinhält und Kommandos an die Tape Library sendet. Außerdem sollten Tapes so präpariert werden, dass in diese anstelle eines Speichermediums andere Sachen gelegt werden können. Hierzu wurde ein Prototyp angefertigt.

In diesem Bericht werde ich zuerst auf den Aufbau der verwendete Tape Library eingehen. Anschließend beginne ich das geschriebene Programm zu erläutern. Dafür werde ich mit den von der Tape Library unterstützten SCSI-Kommandos beginnen. Zu diesen werde ich im Einzelnen aufführen was diese Kommandos bewirken und ob diese erfolgreich Implementiert wurden. Anschließend erläutere ich den Teil der Visualisierung des Programmes. Dazu gehören der Aufbau und die Funktionsweise. Im Anschluss wird dargestellt, wie die Kommunikation mit der Tape Library hergestellt wurde bzw. wie die SCSI-Kommandos versendet wurden. Zum Schluss des Berichtes werde ich etwas zum Bau des Tape-Prototyps schreiben.

Zunächst schreibe ich noch etwas zu meiner Motivation für dieses Projekt. Die Programmiersprache Python war mir zu Projektbeginn gänzlich unbekannt. Das Lernen dieser Sprache stellte für mich zunächst eine große Herausforderung, aber auch gleichzeitig eine große Motivation dar. Die Bastelarbeit, welche ich an dem Tape vornehmen durfte, bot eine gute Ausgleichsmöglichkeit zu der Programmierarbeit. Diese wurde leider im Großteil der aufgewendeten Zeit vernachlässigt, so dass der Ausgleich nicht richtig stattgefunden hat. Außerdem sehe ich in diesem Projekt eine Chance, mein restliches Studium mit der Informatik in Verbindung zu bringen. Ich studiere Lehramt an beruflichen Schulen mit der Fachrichtung Chemotechnik. Daher liegt der Kern meines Studiums in der Chemie und die Informatik findet wenig Bezug dazu. Durch die Bastelarbeit lässt sich *theoretisch* alles in die Tapes legen. Wenn man in die Tapes zum Beispiel verschie-

dene Salze legen würde, könnte man aus einer Tape Library eine Chemikalienausgabe gestalten. Dies würde die Arbeit im Labor vereinfachen, da nicht jede Chemikalie per Hand raus gesucht werden müsste. Beachtet werden müsste bei dieser Verwendungsart aber, dass es sich lediglich für feste Substanzen eignen würde. Flüssigkeiten, wie Säuren oder Laugen, sollten nicht in der Library verwahrt werden, da das Risiko der Beschädigung hier hoch liegen würde.

2 Aufbau der Tape Library StorageTek L180



1. 84-cartridge-cell configuration
2. 140-cartridge-cell configuration
3. 174-cartridge-cell configuration
4. Blocked storage cells (no cartridges permitted)
5. Swap cell
6. Cells reserved for cleaning or diagnostic cartridges
7. CAP with two magazines of five cartridges
8. Reserved cell array with a swap cell and cells reserved for cleaning and diagnostic cartridges
9. Drive column (shown with DLT, SDLT or LTO Ultrium drives installed)

Abb. 1: Schematischer Aufbau der Tape Library StorageTek L180 (vgl. StorageTek - *General Information Manual*, 2005, S. 3-2).

In Abbildung 1 ist der schematische Aufbau der Tape Library StorageTek L180 dargestellt. Es gibt die Library in drei Ausführungen. In der 84-, 140- und 174 –

Storage Slot Variante. In diesem Projekt wurde die 84 Slot Variante verwendet. Wenn man von vorne in die Library rein guckt befindet sich ganz Links eine Reihe mit 28 Storage Slots, in welchen Tapes liegen können. Rechts daneben befindet sich die installierten Tape Drives. In der L180 können maximal zehn Tape Drives installiert werden, bei der verwendeten Library waren es zwei. Auf der rechten Seite der Tape Drives befindet sich zwei weitere Reihen mit je 28 Storage Slots. Außerhalb der Library befindet sich ganz oben das Operator Panel, auf welchem verschiedene Informationen stehen und an welchem man mit verschiedenen Tasten agieren kann. Rechts an der Seite ist die CAP zu sehen, welche sich über das Operator Panel öffnen lässt. Öffnet man diese, gelangt man an die zwei Magazine der Input/Output – Slots, welche je fünf Slots beherbergen und von innen gesehen rechts neben der dritten Reihe Storage Slots liegen. In der Mitte der Library befindet sich ein Roboterarm, welcher an jede Stelle der inneren Library gelangt. Außerdem gibt es einen Block von sechs Tape Slots, welche für Säuberungs- und Diagnosetapes reserviert ist. Diese wurden, ebenso wie die Swap cell, während des Projektes und beim Programmieren des Programmes nicht berücksichtigt.

3 Das Programm

3.1 Unterstützte Kommandos

In Tabelle 1 sind alle von der Tape Library StorageTek L180 unterstützen SCSI-Kommandos aufgeführt.

Tab. 1: Unterstützte SCSI-Kommandos der Tape Library StorageTek L180 (vgl. StorageTek L180/L700e - Tape Libraries and PTP, 2007, Kap. 6).

Initialize Element Status	Initialize Element Status with Range
Inquiry	Log Sense
Mode Select	Mode Sense
Move Medium	Persistent Reserve In
Persistent Reserve Out	Position to Element
Prevent/Allow Medium Removal	Read Element Status
Release	Report LUNs
Request Sense	Request Volume Element Address
Reserve	Rezero Unit
Send Diagnostic	Send Volume Tag
Test Unit Ready	Write Buffer

Im Folgenden werde ich zu jedem der in Tabelle 1 aufgeführten SCSI-Kommandos schreiben, was dieses bewirkt und ob es erfolgreich Implementiert wurde.

3.1.1 Initialize Element Status

Das SCSI-Kommando *Initialize Element Status* wird üblicherweise verwendet, um einen Status der Tapes zu erhalten, welche in der Tape Library platziert sind.

Die Tape Library StorageTek L180 akzeptiert dieses Kommando zur Kompatibilität, führt darauf allerdings keine Aktion aus. Diese Tape Library verfügt über das Kommando *Read Element Status*, welches das gewünschte Resultat erbringt (vgl. StorageTek L180/L700e - Tape Libraries and PTP, 2007, Kap. 6-5).

Das SCSI-Kommando Initialize *Element Status* wurde daher nicht implementiert.

3.1.2 Initialize Element Status with Range

Das SCSI-Kommando *Initialize Element Status with Range* wird üblicherweise verwendet, um einen Status der Tapes zu erhalten, welche in einem bestimmten, festlegbaren Bereich in der Tape Library platziert sind.

Die Tape Library StorageTek L180 akzeptiert dieses Kommando zur Kompatibilität, führt darauf allerdings keine Aktion aus. Diese Tape Library verfügt über das Kommando *Read Element Status*, welches das gewünschte Resultat erbringt (vgl. StorageTek L180/L700e - Tape Libraries and PTP, 2007, Kap. 6-6).

Das SCSI-Kommando Initialize *Element Status with Range* wurde daher nicht implementiert.

3.1.3 Read Element Status

Das SCSI-Kommando *Read Element Status* wird verwendet, um von der Library Informationen über den Status ihrer verschiedenen Elemente zu erhalten. Dabei lassen sich verschiedene Optionen wählen. Man kann Informationen über alle Elemente, nur über den Roboterarm, die Storage Elemente, die Import/Export-Elemente und über die Tape Drives erhalten. Zu diesen Informationen gehört zum Beispiel, ob sich in dem jeweiligen Tape Slot ein Tape befindet. Sollte sich in dem Slot ein Tape befinden, so wird zu diesem die Seriennummer mit ausgegeben (vgl. StorageTek L180/L700e - Tape Libraries and PTP, 2007, Kap. 6-112).

3.1.4 Inquiry

Das SCSI-Kommando *Inquiry* wird verwendet, um von der Library bzw. allgemein von dem angefragten Gerät Informationen über dieses zu erhalten.

Informationen können zum Beispiel der Typ des Gerätes (hier: ein Medium Changer Device), den Hersteller (hier: „STK“ – StorageTek), die Modellbezeichnung (hier: „L180“), oder das Product Revision Level (vgl. StorageTek L180/L700e - Tape Libraries and PTP, 2007, Kap. 6-8 – 6-13). Außerdem gibt es die Möglichkeit Informationen zur Seriennummer des Gerätes zu erhalten (vgl. StorageTek L180/L700e - Tape Libraries and PTP, 2007, Kap. 6-15).

Das SCSI-Kommando *Inquiry* wurde implementiert und funktioniert.

3.1.5 Log sense

Das SCSI-Kommando *Log Sense* wird verwendet, um Error Logs von der Library zu erhalten. Dabei gibt es verschiedene Pages, welche man sich ausgeben lassen kann. Zunächst gibt es die *Supported Pages Format* Page. Diese gibt einem aus, welche Pages von der Library ausgegeben werden können und welchen Page Code man in dem Command Descriptor Block (im Folgenden CDB, hierzu später mehr) verwenden muss.

Weiter gibt es die *Last n Errors Events Page Format* Page. Diese gibt einem die zuletzt hinzugefügten Error Logs aus. Jeder Error Log beinhaltet einen Zeitstempel (Jahr – Monat – Tag – Stunde –Minute - Sekunde, einen Symptom-Code und die Zahl, wie oft dieser Fehler bereits aufgetreten ist.

Zuletzt gibt es die *TapeAlert* Page. Diese Seite gibt einem Fehler der Tapes aus, welche beim lesen/schreiben aufgetreten sind. Für jedes Tape gibt es dabei 64 mögliche Fehlerursachen welche alle entweder True oder False sein können. Jede Fehlermöglichkeit wird zurückgegeben (vgl. StorageTek L180/L700e - Tape Libraries and PTP, 2007, Kap. 6-27 – 6-34).

Das SCSI-Kommando *Log Sense* wurde implementiert und funktioniert.

3.1.6 Request Sense

Das SCSI-Kommando *Request Sense* wird verwendet, um von der Library *Sense Data* zu erhalten, welche erstellt wird, wenn Fehlermeldungen auftreten (vgl. StorageTek L180/L700e - Tape Libraries and PTP, 2007, Kap. 6-158).

Das SCSI-Kommando *Request Sense* wurde auf Grund von mangelnder Zeit nicht implementiert. Zur Fehlerausgabe wurde das SCSI-Kommando *Log Sense* implementiert.

3.1.7 Mode Sense

Das SCSI-Kommando *Mode Sense* wird verwendet, um die Betriebs Parameter von der Library zu erhalten. Auch hier gibt es verschiedene Pages, welche ausgegeben werden können. Als für dieses Projekt besonders wichtig, hat sich die *Element Address Assignment* Page herausgestellt. Über diese erhält man die in der Library gespeicherten Adressen für die verschiedenen Elemente. Dazu gehören die Adresse des ersten *Medium Transport Element*s (Roboterarm), des

ersten *Storage Elements* (die Tape Slots in der Tape Library), des ersten *Import/Export Elements* (Tape Slots in der Kappe an der Library zum reinlegen und rausnehmen von Tapes, im Folgenden CAP) und die Adresse des ersten *Data Transfer Elements* (in der Library eingebaute und angeschaltete Tape Drives). Außerdem lässt sich zu jeder eben genannten Elementgruppe noch die Anzahl der gesamten Elemente der jeweiligen Gruppe ausgeben. So ist es möglich, die jeweils weiteren Adressen der Elemente zu berechnen (vgl. StorageTek L180/L700e - Tape Libraries and PTP, 2007, Kap. 6-61 – 6-68).

Das SCSI-Kommando *Mode Sense* wurde implementiert und funktioniert.

3.1.8 Mode Select

Das SCSI-Kommando *Mode Select* wird verwendet, um die Werte der in Abschnitt 2.1.5 bereits erwähnten Betriebsparameter der Library zu verändern. Diese werden von der Library zum Beispiel beim Starten des Systems verwendet, um sich selbst zu konfigurieren. Bevor die Library diese Parameter „blind“ verändert, überprüft sie zunächst, ob der gewünschte Parameter valide ist. Sollte er es nicht sein, wird der Parameter nicht geändert (vgl. StorageTek L180/L700e - Tape Libraries and PTP, 2007, Kap. 6-35).

Das SCSI-Kommando *Mode Select* wurde implementiert, wird aber von dem Programm nicht verwendet.

3.1.9 Move Medium

Das SCSI-Kommando *Move Medium* wird verwendet, um ein Tape von einem spezifischen Tape Slot zu einem anderen spezifischen Tape Slot zu transferieren. Im ersten Tape Slot muss sich ein Tape befinden, in dem zweiten darf sich kein Tape befinden. Sollte eine dieser beiden Bedingungen verletzt werden, gibt die Library eine Fehlermeldung zurück (vgl. StorageTek L180/L700e - Tape Libraries and PTP, 2007, Kap. 6-98).

Das SCSI-Kommando *Move Medium* wurde implementiert und funktioniert.

3.1.10 Position To Element

Das SCSI-Kommando *Position To Element* wird verwendet, um den Roboterarm zu einem spezifischen Element zu bewegen (vgl. StorageTek L180/L700e - Tape Libraries and PTP, 2007, Kap. 6-109).

Das SCSI-Kommando *Position To Element* wurde implementiert und funktioniert.

3.1.11 Persistent Reserve Out

Das SCSI-Kommando *Persistent Reserve Out* wird verwendet, um die Library für einen bestimmten Anwender für eine bestimmte Aktion zu reservieren, wieder freizugeben, oder alle Reservierungen zu löschen. Das Kommando wird nur verwendet, wenn mehrere User auf eine Library zugreifen (vgl. StorageTek L180/L700e - Tape Libraries and PTP, 2007, Kap. 6-105).

Das SCSI-Kommando *Persistent Reserve Out* wurde nicht implementiert, da auf die Library lediglich von einem Gerät zugegriffen wird.

3.1.12 Persistent Reserve In

Das SCSI-Kommando *Persistent Reserve In* wird verwendet, um von der Library zu erfahren, welche Anwender fehlerhafte, oder in Konflikt stehende persistente Reservierungen besitzen (vgl. StorageTek L180/L700e - Tape Libraries and PTP, 2007, Kap. 6-100).

Das SCSI-Kommando *Persistent Reserve In* wurde nicht implementiert, da auf die Library lediglich von einem Gerät zugegriffen wird.

3.1.13 Reserve

Das SCSI-Kommando *Reserve* wird verwendet, um verschiedene Reservierungen der Library vorzunehmen. Es gibt zum einen die Möglichkeit die Library als Ganzes zu reservieren, zum anderen kann man bestimmte Elemente der Library reservieren. Die Library unterstützt dabei maximal 64 Elementreservierungen (vgl. StorageTek L180/L700e - Tape Libraries and PTP, 2007, Kap. 6-175).Das SCSI-Kommando *Reserve* wurde nicht implementiert, da auf die Library lediglich von einem Gerät zugegriffen wird.

3.1.14 Release

Das SCSI-Kommando *Release* wird verwendet, um Library- oder Elementreservierungen aufzuheben, welche zuvor über ein *Reserve* – Kommando vorgenommen wurden. Dabei kann nur der Anwender, welche die Reservierung vorgenom-

men hat, das *Release* – Kommando verwenden und damit die Reservierung aufheben. Weitere Anwender können das *Release* – Kommando zwar auch anwenden, allerdings wird in diesem Fall die Reservierung nicht aufgehoben (vgl. StorageTek L180/L700e - Tape Libraries and PTP, 2007, Kap. 6-155).

Das SCSI-Kommando *Release* wurde nicht implementiert, da auf die Library lediglich von einem Gerät zugegriffen wird.

3.1.15 Prevent/Allow Medium Removal

Das SCSI-Kommando *Prevent/Allow Medium Removal* wird verwendet, um den Zugriff auf die CAP zu erlauben, oder zu verbieten. Wenn das Kommando ausgeführt wird und das Öffnen der CAP verboten wird, erscheint auf dem Operator Panel ein „locked“ (vgl. StorageTek L180/L700e - Tape Libraries and PTP, 2007, Kap. 6-110).

Das SCSI-Kommando *Prevent/Allow Medium Removal* wurde implementiert und funktioniert.

3.1.16 Report LUNs

Das SCSI-Kommando *Report LUNs* wird verwendet, um von der Library ihre Logical Unit Number (LUN) zu erhalten, an welche der Anwender Kommandos senden kann (vgl. StorageTek L180/L700e - Tape Libraries and PTP, 2007, Kap. 6-156).

Das SCSI-Kommando *Report LUNs* wurde nicht implementiert, da die Library ausschließlich die LUN 0 wiedergibt.

3.1.17 Send Volume Tag

Das SCSI-Kommando *Send Volume Tag* wird verwendet, um mit Hilfe der Library das Volume Tag eines Tapes zu überschreiben. In dem Volume Tag eines Tapes ist zum Beispiel festgelegt, ob was für eine Art Tape es sich handelt (zum Beispiel Cleaning Tapes; vgl. StorageTek L180/L700e - Tape Libraries and PTP, 2007, Kap. 6-182).

Das SCSI-Kommando *Send Volume Tag* wurde nicht implementiert. Es wurde mit der Implementation begonnen und es gibt eine Datei für den CDB von diesem Kommando. Allerdings ist es etwas gefährlich dieses Kommando zu verwenden, da die Library in Folge dessen die Tapes eventuell nichtmehr korrekt erkennt.

3.1.18 Request Volume Element Address

Das SCSI-Kommando *Request Volume Element Address* wird verwendet, um von der Library das Resultat eines vorherigen *Send Volume Tag* – Kommandos zu erhalten (vgl. StorageTek L180/L700e - Tape Libraries and PTP, 2007, Kap. 6-171).

Das SCSI-Kommando *Request Volume Element Address* wurde nicht implementiert, da das SCSI-Kommando *Send Volume Tag* nicht implementiert wurde.

3.1.19 Rezero Unit

Das SCSI-Kommando *Rezero Unit* übt keinerlei Aktionen aus. Die Library akzeptiert dieses Kommando aus Kompatibilitätsgründen (vgl. StorageTek L180/L700e - Tape Libraries and PTP, 2007, Kap. 6-178).

Das SCSI-Kommand *Rezero Unit* wurde daher nicht implementiert.

3.1.20 Send Diagnostic

Das SCSI-Kommando *Send Diagnostic* wird verwendet, um die Library eine Selbstdiagnose ausführen zu lassen. Dabei Kalibriert die Library sich neu. Während die Aktion ausgeführt wird, trennt die Library ihre Verbindung vom Zugriffsgerät. Die Library gibt ihren Status wieder, welcher auf dem Resultat dieser Aktion beruht (vgl. StorageTek L180/L700e - Tape Libraries and PTP, 2007, Kap. 6-179).

Das SCSI-Kommando *Send Diagnostic* wurde nicht vollständig implementiert. Beim ausführen dieses Kommandos beginnt die Library sich selbst zu kalibrieren, bricht aber nach kurzer Zeit den Vorgang ab. Der Fehler liegt vermutlich daran, dass die Rückgabemöglichkeit des Status nicht korrekt implementiert wurde.

3.1.21 Test Unit Ready

Das SCSI-Kommando *Test Unit Ready* wird verwendet, um herauszufinden, ob die Library angeschaltet ist und bereit ist weitere SCSI-Kommandos zu empfangen. Die Library gibt nur dann eine Fehlermeldung zurück, wenn die Library nicht bereit ist (vgl. StorageTek L180/L700e - Tape Libraries and PTP, 2007, Kap. 6-185).

Das SCSI-Kommando *Test Unit Ready* wurde implementiert und funktioniert.

3.1.22 Write Buffer

Das SCSI-Kommando *Write Buffer* wird verwendet, um den Mikrocode der Library zu updaten. Eine Sequenz von einem oder mehreren *Write Buffer* – Kommandos werden Download genannt. Hat ein Download fehlerfrei stattgefunden wird der neue Mikrocode auf den Flashspeicher geschrieben und die Library resetet (vgl. StorageTek L180/L700e - Tape Libraries and PTP, 2007, Kap. 6-186).

Das SCSI-Kommando wurde nicht implementiert, da es gefährlich sein kann dieses zu verwenden. Bei fehlerhafter Implementation, oder fehlerhaftem neu aufgespielten Mikrocode kann es zu schweren Schäden an der Library kommen.

3.1.23 Load/Unload

Außer für die Tape Library StorageTek L180 wurde das SCSI-Kommando *Load/Unload* für IBM Tape Drives verwendet. Dieses wird verwendet um ein Tape in ein Tape Drive zu laden, oder es auszuwerfen (vgl. IBM Corp.: SCSI-Reference, 2007, S. 45), so dass der Roboterarm der Tape Library dieses wieder entfernen kann.

3.2 Die Visualisierung

3.2.1 Allgemeine Informationen und Starten des Programms

Das Programm wurde in der Sprache Python für das Betriebssystem Linux geschrieben. Während des Projektes wurde hierbei Ubuntu verwendet. Zum Starten des Programms navigiert man zunächst über die Shell in den Ordner, in welchem sich das Python-Script *tape-vis* befindet. Um das Programm in der reinen Visualisierungsvariante zu starten muss „python tape-vis“ eingegeben und bestätigt werden. Ist die Library mit dem Rechner verbunden und man möchte das Programm so starten, dass Kommandos an die Library gesendet werden können, so muss „python tape-vis l180 --real“ eingegeben und bestätigt werden. Wenn „python tape-vis -h“ eingegeben wird, erscheint eine Hilfeanzeige. Das gleiche geschieht, wenn nach „python tape-vis“ noch weitere Zeichen hinzugefügt werden, welche nicht „--real“ sind.

Die grafische Oberfläche mit den grundlegenden Funktionen wurde mir von Dr. Julian Kunkel zur Verfügung gestellt.

Im nächsten Abschnitt wird der optische Aufbau des Programmes erläutert. Darauf folgt die Funktionsweise dessen.

3.2.2 Das User Interface

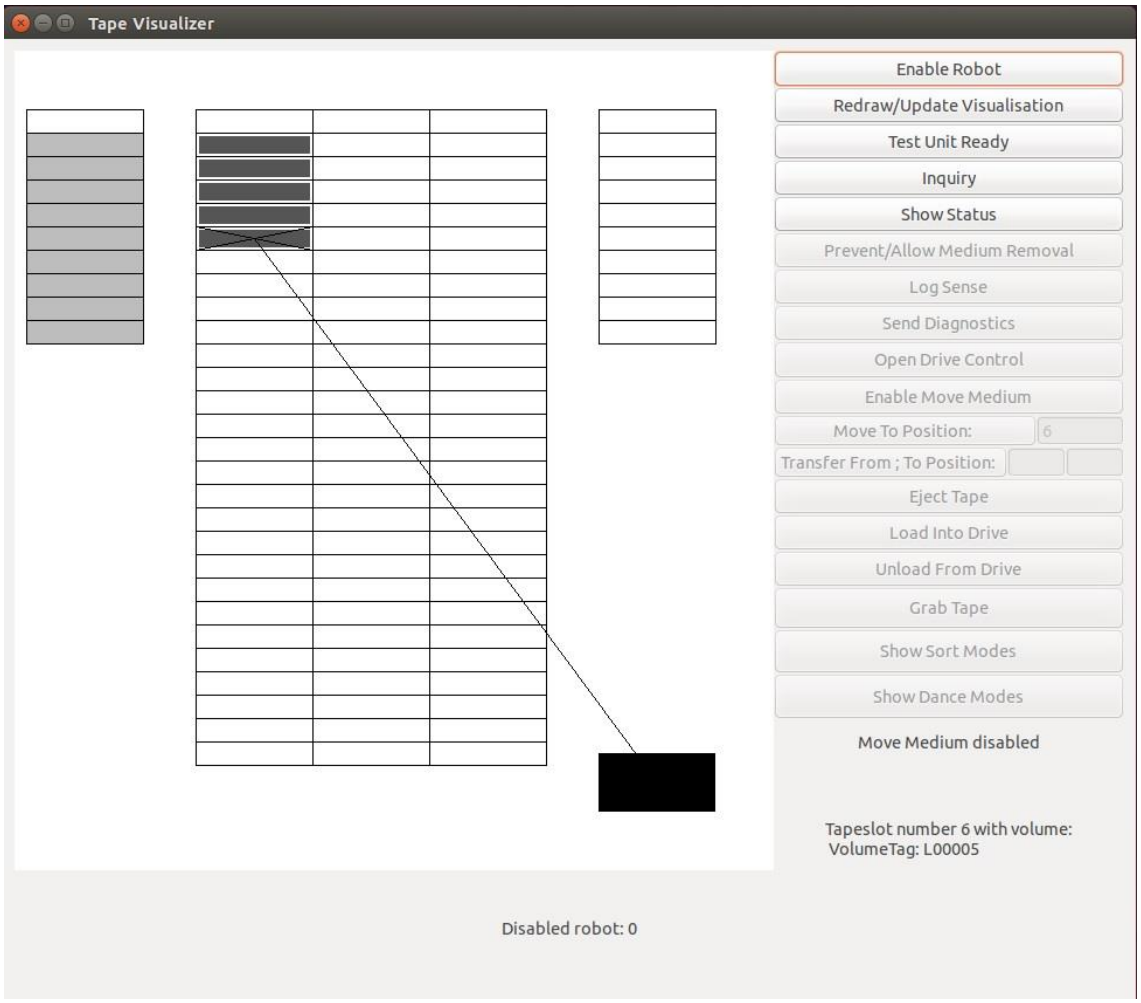


Abb. 2: Gestartetes Programm.

In Abbildung 2 ist das gestartete Programm dargestellt. Das User Interface (im Folgenden UI) wurde mit dem Modul PyGTK erstellt (vgl. Homepage von PyGTK). In dem großen weißen Feld findet die eigentliche Visualisierung des Systems statt. Auf der linken Seite befinden sich die zehn möglichen Tape Drives. Da in der Library maximal zehn Tape Drives installiert werden können, werden alle Tape Drives, welche nicht installiert sind, ein gegraut. In diesem Fall ist ein Tape Drive installiert. Rechts daneben befinden sich drei Reihen zu je 28 Storage Slots. In diesen können sich dünne schwarze Blöcke befinden, welche die Tapes repräsentieren. Rechts neben den Storage Slots befinden sich die zehn Input/Output – Slots der CAP. Unten Rechts ist ein großer schwarzer Block zu sehen. Dieser stellt den Roboter dar. Die schwarze dünne Linie, welche von dem Block ausgeht ist der Roboterarm, an dessen Ende sich die Roboterhand befindet.

Am unteren Rand des Programmfensters befindet sich die Informationsleiste. Auf dieser werden Informationen zu dem aktuellen Status des Programms angezeigt. Zum Beispiel bedeutet „Disabled robot: 0“, dass der Roboter mit der Nummer 0 deaktiviert wurde. Am rechten Rand des Programmfensters befindet sich die Buttonleiste. Unter der Buttonleiste befindet sich eine Anzeige, welche Informationen darüber liefert, ob der Roboter ein Tape, welches sich im aktuellen Tape Slot befindet, automatisch zum nächsten Tape Slot transferiert. Dies ist eine über den Button „Enable Move Medium“ aktivierbare Funktion. Unter dieser Anzeige befindet sich die Tape Slot-Informationsleiste, welche Informationen zu dem Tape Slot anzeigt, über welcher sich der Roboter aktuell befindet. Diese Anzeige wird außerdem verwendet, wenn eine *Inquiry*-Anfrage an das Gerät gesendet wurde, um die Daten anzuzeigen.

Nun werde ich die Buttonleiste noch etwas näher erläutern. Der erste Button mit der Aufschrift „Enable Robot“ bietet die Möglichkeit den Roboter zu aktivieren, so dass dieser bewegt werden kann. Über denselben Button lässt dieser sich auch wieder deaktivieren. Wenn der Roboter aktiviert ist, werden die ganzen Buttons, welche in Abbildung 2 deaktiviert sind, ebenfalls aktiviert. Der nächste Button mit der Aufschrift „Redraw/Update Visualisation“ erstellt die Visualisierung neu. Das ist notwendig, wenn ein Tape mittels der CAP in das System gegeben, bzw. dem System entnommen wurde. Mit dem Button „Test Unit Ready“ kann überprüft werden, ob die Library aktuell einsatzbereit ist. Das Ergebnis wird in der Informationsleiste ausgegeben. Wie in Abbildung 3 zu sehen ist, öffnet der Button „Show Status“ ein zweites Fenster, in welchem die Statusinformationen der einzelnen Tape Slots dargestellt sind. Zu diesen Informationen gehört, ob sich in dem Tape Slot ein Tape befindet und falls ja, welche Seriennummer diese besitzt. Der Button „Log Sense“ ist nur auswählbar, wenn der Rechner mit der Library verbunden ist und das Programm nicht im Simulationsmodus gestartet wird. Mit einem Klick auf diesen öffnet sich eine Auswahlmöglichkeit, welche Art von Fehlermeldungen man lesen möchte (siehe Abschnitt 3.1.5). Nach Auswahl dieser öffnet sich ein neues Fenster mit den gewünschten Informationen. Auch der Button „Send Diagnostics“ ist nur auswählbar, wenn das Programm sich nicht im Simulationsmodus befindet. Per Druck auf diesen, wird ein *Send Diagnostics* – SCSI-Kommando an die Library gesendet (siehe Abschnitt 3.1.20). Der Button „Open Drive Control“ soll nur auswählbar sein, wenn das Programm sich nicht im Simulationsmodus befindet und sowohl die Library, als auch deren Tape Drives an

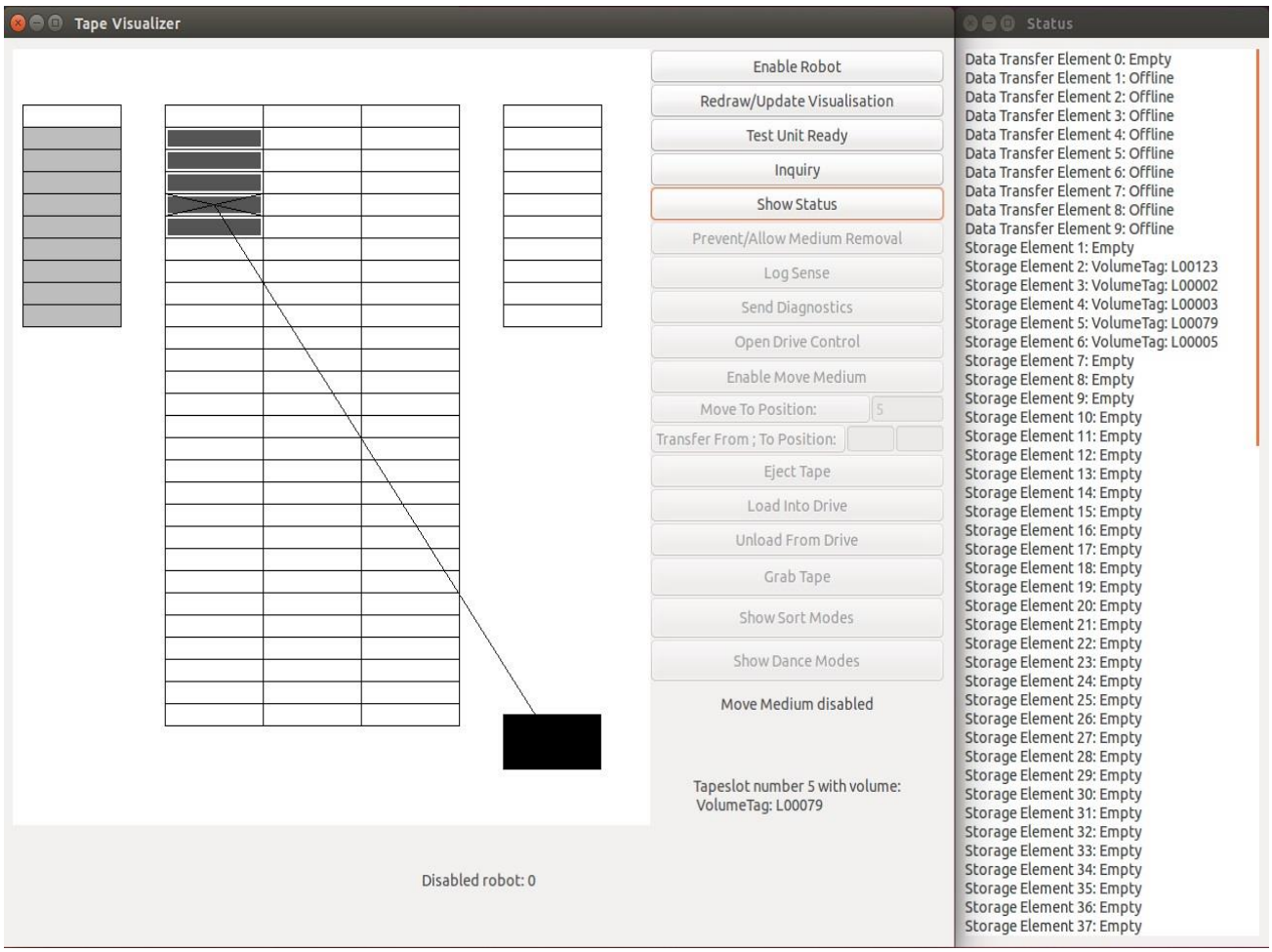


Abb. 3: Gestartetes Programm mit Statusleiste auf der rechten Seite.

den Rechner angeschlossen sind. Da dies zurzeit noch nicht möglich war, wurde der Button zunächst gänzlich deaktiviert und muss im Code aktiviert werden, sobald beide Komponenten angeschlossen werden können. Beim drücken dieses Buttons öffnet sich ein neues Fenster, welches es einem Ermöglicht SCSI-Kommandos direkt an das angeschlossene Tape Drive zu senden und dadurch zum Beispiel ein geladenes Tape auszuwerfen. Der Button „Enable Move Medium“ aktiviert bzw. deaktiviert wie vorher bereits beschrieben die Funktion, dass der Roboter ein Tape aus einem Tape Slot beim drüber fahren direkt mit nimmt und in den Ziel – Tape Slot platziert, sofern dieser frei ist.

Die nächsten beiden Buttons haben direkt mit der Steuerung des Roboters zu tun. Der Button „Move To Position“ ist nur auswählbar, wenn eine gültige Tape Slot-Nummer in das Textfeld rechts neben dem Button eingegeben wurde. Der Roboter bewegt sich daraufhin zu dem angegebenen Tape Slot. Der Button „Transfer From; To Position“ ist nur auswählbar, wenn in beide Textfelder rechts neben dem Button gültige Tape Slot-Nummern eingegeben wurden. Dabei muss

in das linke Textfeld die Nummer eines Tape Slots, in welchem sich ein Tape befindet, in das rechte Textfeld eine Nummer, in welchem sich kein Tape befindet. Sind diese Bedingungen erfüllt, transferiert der Roboter das Tape aus dem ersten Tape Slot in den zweiten Tape Slot.

Die Steuerung des Roboters kann alternativ auch einfach per Klick in die Visualisierung vorgenommen werden. Über einen Klick auf den Roboter lässt sich dieser aktivieren. Ist der Roboter aktiviert, färbt er sich rot. Per Klick auf einen Tape Slot bewegt sich der Roboter zu diesem. Ist die Funktion „Move Medium“ aktiviert, überprüft der Roboter beim vor dem weiterbewegen, ob in dem aktuellen Tape Slot ein Tape liegt. Ist dies der Fall wird der Roboter, wie bereits beschrieben, dieses mitnehmen und in dem Ziel-Tape Slot platzieren.

Der Button „Eject Tape“ kann verwendet werden, um ein Tape direkt in einen I/O-Slot zu transferieren. Dabei muss der Roboter sich über einem Tape Slot mit Tape befinden. Das Tape wird in den ersten freien I/O-Slot transferiert. Der Button „Load Into Drive“ kann verwendet werden, um ein Tape direkt in ein Tape Drive zu transferieren. Auch hierbei muss sich der Roboter über einem Tape Slot mit Tape befinden. Das Tape wird in das erste Tape Drive transferiert, welches kein Tape beinhaltet. Der Button „Unload From Drive“ hingegen kann verwendet werden, um ein Tape direkt aus einem Tape Drive in einen Storage Slot zu transferieren. Dabei darf das Tape in dem Tape Drive nicht geladen sein. Es wird in den ersten freien Storage Slot transferiert. Der Sinn des Buttons „Grab Tape“ ist, dass der Roboter sich ein Tape nimmt und dieses bei sich behält, bis man ihm erlaubt es in einen Tape Slot abzulegen. Die Library StorageTek L180 unterstützt dieses Feature nicht, so dass der Button für diese Library deaktiviert wurde.

Die Buttons „Show Sort Modes“ und „Show Dance Modes“ öffnen wie in Abbildung 4 zu sehen ist je ein neues Fenster mit verschiedenen weiteren Möglichkeiten. Der Button „Random Sort“ platziert alle Tapes in der Tape Library an willkürliche, neue Positionen. Der Button „Sort According To Serial“ wird verwendet, um alle Tapes nach ihrer Seriennummer kompakt zu sortieren. Die Sortierung verläuft von der kleinsten Seriennummer zur größten, die Buchstaben in der Seriennummer werden dabei nicht berücksichtigt. Die Platzierung der Tapes beginnt in dem ersten Storage Slot. Der Button „Compackt Sort“ wird verwendet, um die Tapes kompakt zu sortieren. Auch hier beginnt die Platzierung der Tapes beim ersten Storage Slot. Jedoch gibt es hier keine Reihenfolge bei der Sortierung.

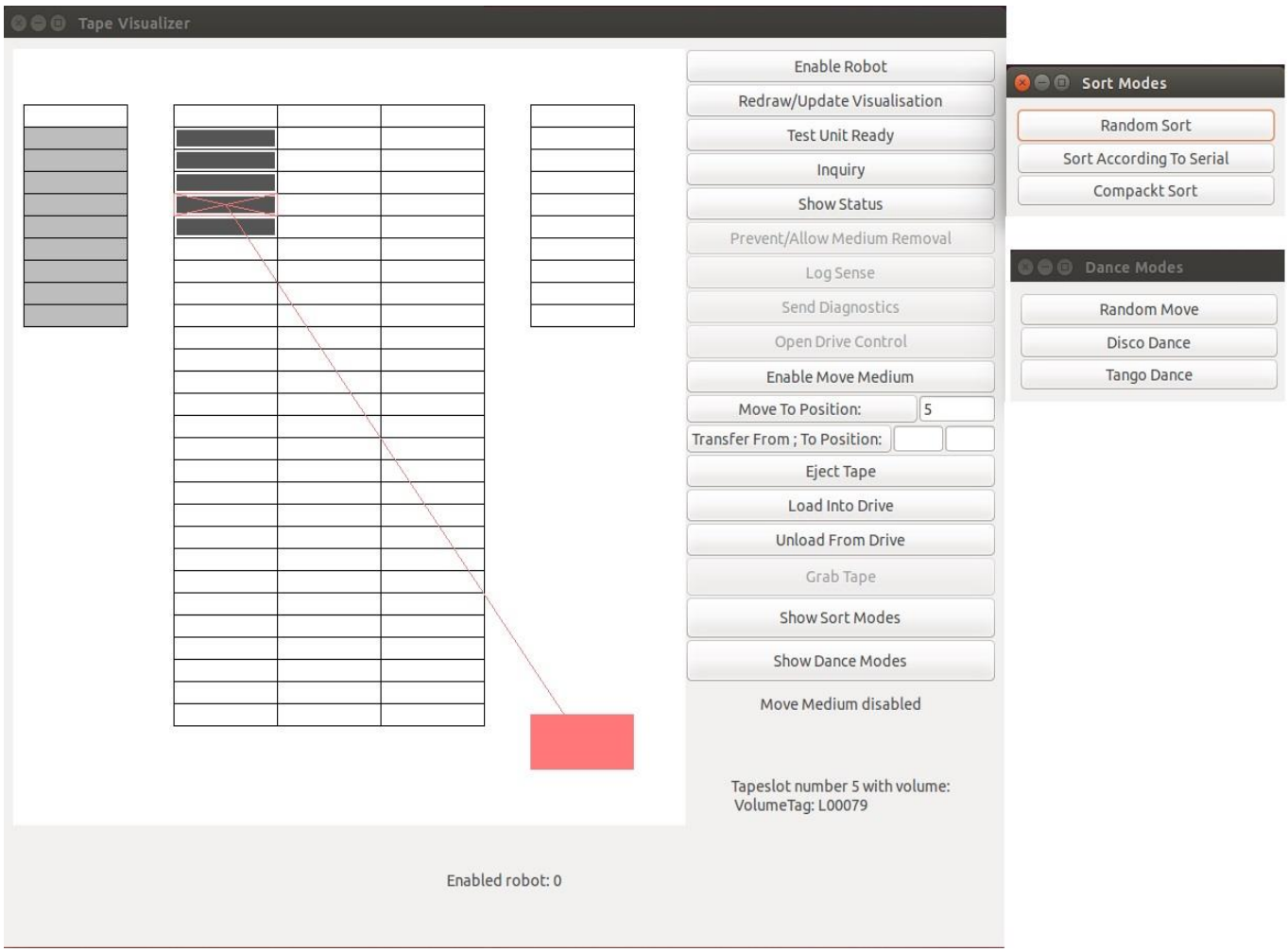


Abb. 4: Gestartetes Programm mit angezeigten Sort- und Dance Modes.

Beim Drücken des Button „Random Move“ beginnt der Roboterarm sich zufällig zu verschiedenen Slots zu bewegen, und so eine Art Tanz auszuführen. Mit dem Button „Disco Dance“ wird eine Art Diskotanz ausgelöst. Der Roboterarm bewegt sich hierbei je zwei Mal nach Oben und Unten, anschließend zwei Mal nach Links und rechts. Beim Drücken des Button „Tango Dance“ bewegt der Roboterarm sich in einer Art Tangoformation. Damit dieses Feature seine Wirkung vollständig entfalten kann, muss der Anwender eine Tangomelodie im Kopf haben, oder Summen, welche im Takt zu den Bewegungen des Roboters passt.

Bei Aktivierung eines Tanzmodus wird die Funktion „Move Medium“ automatisch deaktiviert.

3.2.3 Aufbau und Funktionsweise des Programms

In Abbildung 5 ist ein Klassendiagramm des Programms abgebildet, um dessen Aufbau darzustellen.

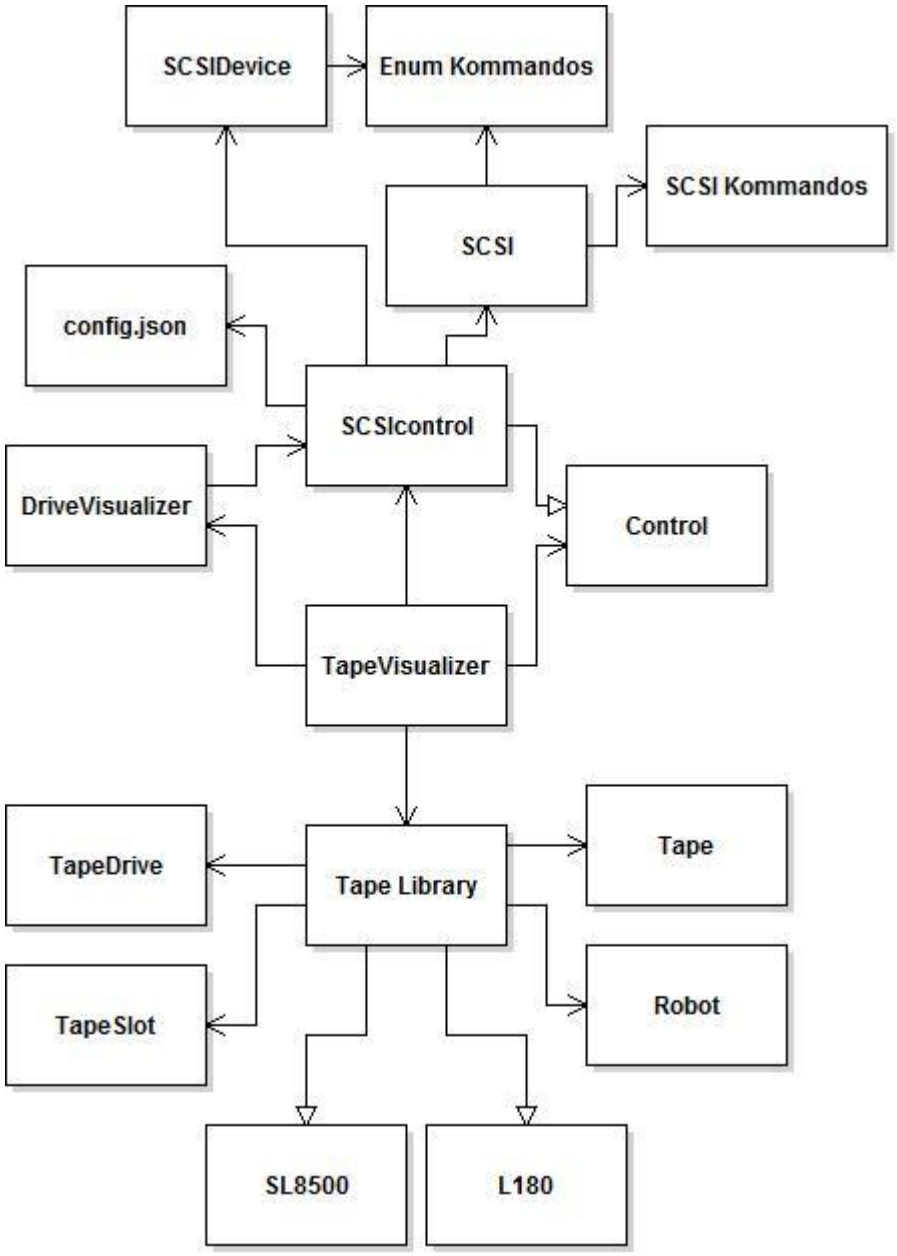


Abb. 5: Klassendiagramm des Programms.

In Abbildung 5 ist zu erkennen, dass alles von der Klasse *TapeVisualizer* ausgeht. Diese wird beim Starten des Programms ausgeführt. Zunächst wird ein Control - Objekt erzeugt. Wenn das Programm ohne richtige Library gestartet wird, handelt es sich dabei um ein Objekt der Klasse *Control*, mit Library um ein

Objekt der Klasse *SCSIcontrol*. Vor dem Starten der UI wird von dem Control - Objekt erfragt, welche Drives aktiv sind und in welchen Slots sich welche Tapes befinden. Wie genau das Erzeugen und Verwenden des *SCSIcontrol* – Objektes funktioniert, wird in Abschnitt 3.3 genauer erläutert. Außerdem wird festgelegt, mit welcher Art Library das Programm gestartet werden soll. Diese Parameter werden bei der Erzeugung des *TapeVisualizer* Objektes übergeben. Im Konstruktor werden nun alle weiteren globalen Objekte und Variablen festgelegt. Es wird ein Objekt der *TapeLibrary* erstellt, welches in der Visualisierung die Library repräsentiert. An diesem Objekt wird nun eingestellt, in welchem Slot welche Tapes liegen und welche Drives verfügbar sind. Anschließend wird am *TapeVisualizer* die *main* – Methode aufgerufen. Hier wird nun die UI erstellt, wie es in Abbildung 2 zu sehen ist. Die Visualisierung selbst wird dabei in einer *gtk.DrawingArea* erzeugt, welche in eine *gtk.EventBox* eingebettet wird. Zur weiteren Gestaltung der ganzen Buttons und Textfelder werden weitere horizontale *gtk.HBox* und vertikale *gtk.VBox* verwendet. Wenn alle Interfaceobjekte erstellt wurden, werden diese mit der Methode *show_all()* sichtbar gemacht.

Bevor ich weiter auf die Klasse *TapeVisualizer* eingehe, fahre ich mit der Klasse *TapeLibrary* fort. Diese erzeugt bei der Erstellung Listen von allen Robotern, Drives, Tape Slots und I/O – Slots, welche in der Library vorhanden sind. Dies wurde für die Library L180 optimiert. Zu Testzwecken wurde ein Prototyp der SL8500 ebenfalls implementiert, in der weiteren Programmentwicklung aber nicht explizit berücksichtigt. In der Klasse *TapeLibrary* werden im Anschluss mit den Methoden *initializeDrives* und *initializeTapes* die Verfügbaren Drives kenntlich gemacht und in die entsprechenden Tape Slots Tapes eingefügt. Außerdem wird in dieser Klasse bestimmt, welcher Tape Slot angeklickt wurde, welche Nummer ein spezifischer Tape Slot hat, welcher Tape Slot zu welcher Nummer gehört und welcher Roboter angeklickt wurde. Es lassen sich weiterhin überprüfen, ob ein bestimmtes Tape verschoben werden darf. Dem Roboter können befehle gegeben werden, wie sich einfach nur zu bewegen, oder ein Tape zu transferieren. Dafür kann auch bestimmt werden, welcher Roboter für den gewünschten Befehl überhaupt zuständig ist. Dies ist wichtig, wenn es mehr als einen Roboter in der Library gibt. In dieser Klasse kann auch der Status der Tape Slots abgefragt werden.

Weiter geht es mit der schon oft erwähnten Klasse *Robot*. Jedes Objekt der Klasse *Robot* besitzt eine Nummer, über welche dieser identifiziert werden kann. Er besitzt zudem eine Liste *(_tapesSelected*) aller Tape Slots, zu welchen er sich bewegen soll und einen Verweis auf das Tape, welches er aktuell hält, sofern er eines hält. Wenn in der Visualisierung auf einen Tape Slot geklickt wird als Zeichen, dass sich der Roboter zu diesem bewegen soll, so wird dieser über die *TapeLibrary* zu der *_tapesSelected* Liste des Roboters hinzugefügt. Die Hauptfunktion des Roboters läuft nun in der Methode *timestep* ab. Im *TapeVisualizer* wird die erste Methode *timestep* erstellt, welche Zeitschritte erstellt und in jedem Zeitschritt Funktionen ausführen kann. Unter Anderem wird die Methode *timestep* in der *TapeLibrary* aufgerufen, welche wiederum die *timestep* – Methode des Roboters aufruft. Zuerst wird nun überprüft, ob die *_tapesSelected* – Liste größer als 0 ist. Dies bedeutet, dass es noch weitere Tape Slots gibt, zu welchen der Roboter sich bewegen soll. Wenn der Roboter zudem auch aktuell kein Ziel hat, wird nun das erste Element der Liste genommen und als neues Ziel deklariert. Wenn der Roboter ein Ziel hat, sich aber noch nicht bei diesem befindet, verändert er seine Position pro Zeitschritt um 30 Pixel in die Richtung des Ziels. Auf diese Weise entsteht die in der Visualisierung sichtbare Bewegung. Wenn der Roboter sein Ziel erreicht hat und er ein Tape hält, welches er ablegen soll, so legt er das Tape nun in den aktuellen Tape Slot ab. Zudem wird der erreichte Tape Slot aus der *_tapesSelected* – Liste entfernt. Sollte sich in dem aktuellen Tape Slot ein Tape befinden, welches nicht gerade erst eingefügt wurde und der Roboter hat die Erlaubnis Tapes zu transferieren, so wird das Tape aus dem aktuellen Tape Slot entnommen. Zum nächsten Zeitschritt wird anschließend nach einem neuen Ziel gesucht.

Nun komme ich zu den etwas kleineren Klassen. Beginnen möchte ich mit der grundlegenden Einheit, dem *Tape.* Jedes Objekt der Klasse *Tape* bekommt eine Seriennummer zugewiesen. Außerdem besitzt es eine Methode um kenntlich zu machen, dass es per Klick im UI ausgewählt wurde.

Zu den grundlegenden Einheiten zählt auch die Klasse *TapeSlot*. Jedes Objekt der Klasse *TapeSlot* besitzt den Namen „TapeSlot“, wodurch dieser von der Klasse *TapeDrive* unterschieden werden kann, da sich diese beiden in ihrer Implementation sehr ähneln. Außerdem besitzt jedes Objekt eine spezifische Num-

mer und einen Verweis auf ein potentielles Tape, welches sich in dem Slot befinden kann. Es gibt hier Methoden, um ein Tape in den Tape Slot einzufügen, oder zu entfernen.

Die Klasse *TapeDrive* gleicht in ihrem Aufbau der Klasse *TapeSlot* sehr. Auch hier gibt es Methoden zum Einfügen und Entfernen von Tapes. Jedes Objekt der Klasse *TapeDrive* besitzt eine eigene Nummer und einen Namen. Der Name ist bei jedem *TapeDrive* „Drive“. Zudem gibt es hier eine Variable, welche anzeigt, ob ein Drive aktiv, oder inaktiv ist. Ist es inaktiv, so wird es in der Visualisierung ein gegraut.

Da nun alle Klassen soweit erläutert wurden, kann ich mit dem *TapeVisualizer* fortfahren. Nach einem Klick in die UI wird hier ermittelt was angeklickt wurde. Handelt es sich um einen Tape Slot und der Roboter darf sich bewegen, so wird die Methode *moveRobot* aufgerufen. Diese lässt sich ebenfalls per Knopfdruck wie in Abschnitt 3.2.2 beschrieben aufrufen. Hier wird zunächst überprüft, ob die Library bereit ist Aktionen auszuführen. Dies geschieht mit dem Kommando *TestUnitReady*. Dazu wird eine entsprechende Methode in dem Control-Objekt aufgerufen, welche einen Boolean zurückgibt. Wenn das Programm im Visualisierungsmodus ausgeführt wird gibt das Control – Objekt immer *True* zurück. Bei Ausführung mit richtiger Library wird hier differenziert. Dazu mehr im Abschnitt 3.3. Sollte die Library nicht bereit sein sollte, wird das Kommando hier abgebrochen. Wenn die Library bereit ist, werden nun die aktuelle Position und die Position des Ziels des Roboters berechnet. Ist dies geschehen, wird überprüft ob die Funktion *Move Medium* aktiv ist. Falls sie aktiv ist und im aktuellen Tape Slot befindet sich ein Tape, so soll der Roboter dieses in das Ziel transferieren. Ist dieses nicht aktiv, soll sich der Roboter einfach nur zu dem Ziel bewegen. Nun wird zuerst versucht die Funktion an der echten Library auszuführen. Wenn *True* zurückgegeben wird, wird der Befehl über ein Objekt der *TapeLibrary* an den Roboter der Visualisierung weitergegeben.

Nach dem gleichen Prinzip funktioniert die Methode *transferMediumTo*. Diese wird aufgerufen, wenn der Roboter aufgefordert wird ein Tape zu transferieren. Auch hier wird zuerst überprüft, ob die Library bereit ist Aktionen auszuführen. Anschließend werden die Start- und Zielpositionen der Tape Slots berechnet und der Befehl wird über das Control-Objekt weitergeleitet. Wenn *True* zurückgegeben wird, wird der Befehl an den Roboter der Visualisierung weitergegeben.

Bei einem Klick auf den Button *Eject* wird die Methode *ejectMedium* aufgerufen, welche die Methode *transferMediumTo* aufruft. Dabei wird als Startslot der aktuelle Slot des Roboters angegeben und als Ziel Slot der erste freie I/O – Slot. Dasselbe gilt für die Methode *loadIntoDrive*, welche als Ziel Slot das erste freie Drive angibt. Bei der Methode *unloadFromDrive* wird als Startslot das erste Tape Drive mit Tape gewählt, als Ziel Slot der erste freie Storage Slot.

Bei der Erzeugung des Objektes des *TapeVisualizers* wird für jeden Tanz- oder Sortiermodus eine boolesche Variable erzeugt und auf *False* gesetzt. Per Knopfdruck auf den entsprechenden Knopf wird die entsprechende Variable auf *True* gesetzt. In der bereits erwähnten *timestep* – Methode wird überprüft, ob sich eine dieser Variablen auf *True* befindet. Wenn dies der Fall ist, so wird die entsprechende Operation durchgeführt, solange bis die Variable per Knopfdruck wieder auf *False* gesetzt wird. Bei den Tanzstilen handelt es sich um eine Abfolge an *moveRobot* befehlen. Bei den Sortierungsmodi um eine Abfolge von gezielten *transferMediumTo* Befehlen. Die Sortierung nach Seriennummer wurde über einen Quick Sort – Algorithmus implementiert.

Bei einem Klick auf *Show Status* wird die Methode *showStatus* ausgeführt. Diese erzeugt mit Hilfe des PyGTK ein neues, scrollbares Fenster. Über das Objekt der Klasse *TapeLibrary* wird der Status der einzelnen Tape Slots abgerufen und in dem Fenster dargestellt.

Mit der Methode *inquiry* wird über das Control – Objekt ein Inquiry Befehl abgesetzt. Läuft das Programm im Simulationsmodus wird der String „Simulation mode“ zurückgegeben.

Die Methoden *mediumRemoval*, welche ein *Prevent/Allow Medium Removal* Befehl absetzt, *Log Sense, sowie sendDiag,* welches ein *SendDiagnostics* Befehl absetzt, können nur aufgerufen werden, wenn das Programm mit einer realen Library gestartet wird.

Es gibt in der Klasse *TapeVisualizer* außerdem Methoden, um die verschiedenen Informationslabels mit verschiedenen Textausgaben zu versehen.

3.3 Kommunikation mit der Tape Library

In Abschnitt 3.2.3 wurde der untere Teil von Abbildung 5 (siehe Seite 17) erläutert. Der obere Teil der Abbildung ist dafür zuständig, die SCSI – Kommandos an die Library zu senden. Beim Umsetzen dieses Teils des Programms hat mir ein Open Source Projekt von Markus Rosjat geholfen, welcher sich damit beschäftigt SCSI – Befehle in Python abzuschicken (vgl. Markus Rosjat, Python-SCSI, 2015).

Wird das Programm mit „-- real" gestartet, so wird als Control – Objekt ein Objekt der Klasse SCSIcontrol erstellt. Diesem wird als Name „TA“ übergeben, damit das Objekt weiß, dass es vom *TapeVisualizer* aufgerufen wurde. Wenn als Name „drive“ übergeben werden würde, wäre das Objekt von der Klasse *DriveVisualizer* aufgerufen worden. In Abbildung 6 ist ein Codeausschnitt aus der Klasse *SCSIcontrol*, welches darstellt was nach dem Erzeugen des Objektes geschieht.

```
20 # init the scsi device path
21       with open('control/config.json') as data_file:
22          data = json.load(data_file)
23       device = "/dev/"
24
25 # if a SCSIcontrol instance is created by an tape archive:
26       if name == "TA":
27          lun =  data["LUN"]
28          device = device + commands.getoutput("ls /sys/bus/scsi/devices/" + lun + "/
   scsi_generic")[0:3]
29          self.device = device
30          self.scsi = SCSI(SCSIDevice(self.device, True))
31          systype = commands.getoutput("cat /sys/bus/scsi/devices/" + lun + "/type")
32          if int(systype) != int(INQUIRY.DEVICE_TYPE.MEDIA_CHANGER_DEVICE):
33             print '%s is not a MediaChanger device' % self.device
34             exit()
35
36          vendor = data["vendor"]
37          sysvendor = commands.getoutput("cat /sys/bus/scsi/devices/" + lun + "/vendor")
38          if sysvendor[0:3] != vendor:
39             print 'The MediaChanger device by %s is not a device by StorageTek' % sysvendor
40             exit()
41
42          model = data["model"]
43          sysmodel = commands.getoutput("cat /sys/bus/scsi/devices/" + lun + "/model")
44          if sysmodel[0:4] != model:
45             print 'The MediaChanger device %s by StorageTek is not the L180 Library' %
   sysmodel
46             exit()
47
48          rev = data["rev"]
49          sysrev = commands.getoutput("cat /sys/bus/scsi/devices/" + lun + "/rev")
50          if sysrev != rev:
51             print 'The L180 TapeLibrary rev %s by StorageTek has the wrong revision level'
   % sysrev
52             exit()
```

Abb. 6: Codeausschnitt aus der Klasse SCSIcontrol.

Als erstes wird der Pfad bestimmt, über welchen die Library zu finden ist. Über eine JSON – Datei wurden Parameter festgelegt, welche die Library eindeutig identifizieren. Diese müssen überprüft werden, damit die Signale nicht versehentlich an ein falsches Gerät gesendet werden. In den Zeilen 23 – 29 wird dieser Pfad bestimmt. Anschließend wird ein Objekt der Klasse *SCSI* erzeugt, welches die SCSI – Kommandos absetzen soll. Diesem wird ein Objekt der Klasse *SCSIDevice* übergeben, welches unsere Library darstellt. In den Zeilen 31 – 34 wird überprüft, ob das angeschlossene Gerät ein Gerät vom Typ Media Changer ist. Dies geschieht indem der Ausgabewert im System, mit dem in einer ENUM - Datei verzeichneten Wert für Media Changer Geräte übereinstimmt. Stimmt dieser Wert nicht überein, handelt es sich nicht um unsere Tape Library und das Programm wird terminiert. In den Zeilen 36 – 40 wird überprüft, ob das Gerät vom richtigen Hersteller gebaut wurde. Dafür wird aus der JSON – Datei der Soll-Hersteller herausgelesen und mit dem Ausgabewert des Systems verglichen. Auch hier wird das Programm beendet, wenn die Werte nicht übereinstimmen. In den Zeilen 42 – 46 wird überprüft, ob das Gerät vom richtigen Modell ist. In den Zeilen 48 – 52 wird überprüft, ob das Gerät die richtige Revisionsnummer besitzt. Bei diesen beiden Punkten wird das gleiche Vorgehen verfolgt, wie bei der Überprüfung des Herstellers. Wurde das Gerät eindeutig verifiziert, wird über ein *Mode Sense* – Kommando die verschiedenen Adressen der Elemente festgelegt (siehe 3.1.7).

Im Folgenden wird anhand des *Log Sense* – Kommandos für die *List n Errors Event Pages* dargestellt, wie ein SCSI – Kommando erstellt wird. Zunächst wird in *SCSIcontrol* über das *SCSI* – Objekt die Methode *libraryErrors()* aufgerufen. Diese ruft ein *Log Sense* Kommando auf und übergibt diesem, dass es den richtigen Page Code für die List n Errors Event Pages verwenden soll. In Abbildung 7 ist dargestellt, wie der Command Descriptor Block (CDB) des *Log Sense* Befehls aufgebaut ist. Ein CDB setzt sich aus mehreren Bytes zusammen. In Abbildung 8 ist ein Codeausschnitt aus dem Log Sense Kommando File dargestellt, welches verdeutlicht, wie die CDB-Bits auf Codeebene implementiert wurde. Die im CDB zu sehenden Werte wurden einzeln aufgeführt. Zudem wird festgelegt, welche Bits des jeweiligen Bytes der jeweilige Punkt einnehmen soll (linke Hexadezimalzahl), sowie in welchem Byte des CDB dieser Punkt vorliegt (rechte Zahl).

Byte	Bit							
	7	6	5	4	3	2	1	0
0	Operation Code (4Dh)							
1	Logical Unit Number			Reserved (0)			PPC (0)	SP (0)
2	PC (1)		Page Code					
3	Reserved (00h)							
4	Reserved (00h)							
5 to 6	(MSB) Parameter Pointer (LSB)							
7 to 8	(MSB) Allocation Length (LSB)							
9	Control (00h)							

Abb. 7: Aufbau des Command Descriptor Blocks (CDB) eines Log Sense SCSI-Kommandos (vgl. StorageTek L180/L700e - Tape Libraries and PTP, 2007, Kap. 6-27).

```
15      _cdb_bits = {'opcode': [0xff, 0],
16                   'ppc': [0x02, 1],
17                   'sp': [0x01, 1],
18                   'pc': [0xc0, 2],
19                   'page_code': [0x3f, 2],
20                   'parameter_pointer': [0xffff, 5],
21                   'allocation_length': [0xffff, 7],}
```

Abb. 8: Codeausschnitt aus dem Log Sense Kommando File.

Anschließend werden die jeweiligen Werte für die Punkte eingetragen (z.B. für den Operation Code 4D, Page Code in diesem Fall 7h), und mit Hilfe eines Converters in ein Bytearray der Größe 10 (der CDB ist 10 Bytes groß) konvertiert. So wird der fertige CDB für das *Log Sense* Kommando erhalten und mit Hilfe der Methode *execute* in der Klasse *SCSIDevice* an die Library geschickt. Dabei schickt die Library Daten zu ihren Errors zurück. Jeder zurückgesendete Error ist dabei 48 Bytes lang (vgl. StorageTek L180/L700e - Tape Libraries and PTP, 2007, Kap. 6-29). Diese müssen noch entschlüsselt werden, um sie in für den Menschen lesbare Form zu bringen. Wie das im Code aussieht ist in Abbildung 9 dargestellt.

```
76      @staticmethod
77      def unmarshall_datain_error_pages(data):
78          """
79          Unmarshall the Log Sense datain with page errors.
80          """
81          result = {}
82          decode_bits(data, LogSense._page_format, result)
83
84          pageLength = result["page_length"]
85          entryCount = pageLength / 48
86          entries = []
87          for p in range(0, entryCount):
88            bytes = data[4+48*p:52+48*p]
89
90            cur = {"faultSymptomCode": str(bytes[0:3]),
91                   "mechanism": str(bytes[4:19]).rstrip(),
92                   "count": int(str(bytes[20:24]),base=16),
93                   "year": int(bytes[24:28]),
94                   "month": int(bytes[28:30]),
95                   "day": int(bytes[30:32]),
96                   "hour": int(bytes[32:34]),
97                   "minute": int(bytes[34:36]),
98                   "second": int(bytes[36:39])}
99            pad = bytes[39:]
100           if len(pad.replace(" ", "")) != 0:
101              print("Error while parsing entry")
102           entries.append(cur)
103
104         result["events"] = entries
105         return result
```

Abb. 9: Codeausschnitt aus dem Log Sense Kommando File. Es wird die Methode unmarshal_datain_error_pages gezeigt.

Zuerst wird ein leeres Set erstellt. Dieses wird einem Converter übergeben, welches die Daten (data) konvertiert und in das Set speichert. In Zeile 84 wird abgefragt, wie viele Zeilen zurückgegeben wurden. Durch die Kenntnis, dass jedes Error Event 48 Bytes lang ist kann berechnet werden, wie viele Error Events zurückgegeben wurden. Anschließend wird jedes Event einzeln dekodiert. Dafür werden lediglich die benötigten Bytes berechnet. In den Zeilen 90 – 98 ist dargestellt, wie das für ein Einzelnes Event aussieht. Die Kenntnis darüber, welcher Byte welche Information enthält ist der Tabelle 6-20 aus dem „StorageTek – Tape Library and PTP“ pdf entnommen. Die dadurch entstehende Menge wird in einer Liste gespeichert, welche am Ende an das *SCSIcontrol* – Objekt zurückgegeben wird. Dieses leitet die Informationen zum *TapeVisualizer* weiter, wo die Daten dem Nutzer präsentiert werden.

4 Das präparierte Tape

Wie zu Beginn erwähnt, war es eine Aufgabe des Projektes ein Tape zu präparieren. Ich habe mich dazu entschieden einen Smartiespender zu bauen. Dieser ist in Abbildung 10 dargestellt.

Abb. 10: Das präparierte Tape. Im oberen Bild ist es von oben zu sehen, im unteren ist es von der Seite mit dem Spenderauslass zu sehen.

Um diesen zu bauen wurde zuerst das Tape der Firma IBM aufgeschraubt und das Innenleben entfernt. Zur Halterung des ursprünglichen Tapes waren im Tape kleine Plastik Erhebungen. Diese wurden mit Hilfe einer Kneifzange soweit möglich entfernt. Nun befand sich in dem Deckel des Tapes ein Loch, welches gestopft werden musste. Dazu wurde ein Stück Pappe in der richtigen Größe ausgeschnitten und mit dem Smarties – Logo beklebt. Dieses Stück Pappe wurde auf ein größeres Stück Pappe geklebt. Dadurch konnte das Logo in das Loch

gesteckt werden und an die Plastikklappe des Tapes geklebt werden. Sollte die Schokolade im Tape warm werden, kann es passieren, dass sie schmilzt. Um einer totalen Verschmutzung vorzubeugen, wurde die Pappe großflächig mit einer Folie überklebt, so dass diese leicht abzuwischen ist.

Das Tape besaß bereits einen Auslass, welcher auch in Abbildung 10 zu sehen ist. Hier wurde ein schwarz angemaltes Stück Streichholz angeklebt, um zum Öffnen etwas Gripp zu geben. Durch diese aufschiebbare Klappe wird das Tape auch befüllt.

5 Zusammenfassung und Future Work

Zusammenfassend lässt sich sagen, dass die Ziele des Projektes weitgehend erreicht wurden. Die Visualisierung funktioniert. Nach einigen Anlaufproblemen wurde auch die Kommunikation mit der Library erfolgreich hergestellt und es funktioniert diese zu steuern. Außerdem wurde ein Tape Prototyp entworfen, in Form eines Smartiespenders. Was noch nicht gut funktioniert ist das SCSI-Kommando *Send Diagnostic*. Dieses bricht immer nach kurzer Zeit ab. Das Problem hier liegt vermutlich daran, dass in der Ausführung des erstellten CDB ein Fehler vorliegt.

Dieser Fehler wäre ein Beispiel für die Future Work. Es können noch fehlende SCSI-Kommandos implementiert werden. Außerdem kann über einen Fibreswitch die Library sowie das Tape Drive mit dem PC verbunden werden. Dadurch kann in den Klasse *TapeVisualizer* der Button zum Starten des *DriveVisualizers* enabled werden. Dafür gibt es im Quelltext eine konkrete Anleitung. Den *DriveVisualizer* kann man auch weiter ausbauen. Es würde sich anbieten ein Kommando zu implementieren, welches die Daten des Tapes liest. Durch schrittweises Auslesen aller Tapes kann man eine Datenbank erstellen, in welcher die Daten der Tapes gespeichert werden. Somit könnten die Daten viel schneller abgerufen werden und es muss nicht jedes Mal das Tape neu ausgelesen werden. Das Programm ist ziemlich auf die Tape Library L180 zugeschnitten, besonders wenn es darum geht die SCSI-Kommandos korrekt abzuschicken. Man könnte das noch etwas offener gestalten, so dass auch weitere Modelle der StorageTek Tape Librarys verwendet werden können.

Außerdem können noch weitere Tapes präpariert werden, beispielsweise gefüllt mir Schokoriegeln.

6 Quellen

[1] IBM Corp.: *IBM Total Storage LTO Ultrium Tape Drive SCSI Reference*, 8. Auflage, USA, 2007.

[2] StorageTek: *L180/L700x/L1400x Tape Libraries – General Information Manual*, Louisville, 2005.

[3] Sun Microsystems, Inc.: *StorageTek L180/L700e Tape Libraries and PTP – Interface Reference Manual*, Kalifornien, 2007.

[4] GitHub Projekt von Markus Rosjat: *python-scsi*, aufzurufen unter: https://github.com/rosjat/python-scsi, letzter Zugriff: 20.11.2015.

[5] Offizielle Homepage von PyGTK, aufzurufen unter: *www.pygtk.org*, letzter Zugriff: 13.11.2015.