Terminal zur Bearbeitung von ARX-Daten mit R

ALEXANDER BEISCHL UND THUY TRAN
Technische Universität München
20. Januar 2017

Zusammenfassung

Morbi tempor congue porta. Proin semper, leo vitae faucibus dictum, metus mauris lacinia lorem, ac congue leo felis eu turpis. Sed nec nunc pellentesque, gravida eros at, porttitor ipsum. Praesent consequat urna a lacus lobortis ultrices eget ac metus. In tempus hendrerit rhoncus. Mauris dignissim turpis id sollicitudin lacinia. Praesent libero tellus, fringilla nec ullamcorper at, ultrices id nulla. Phasellus placerat a tellus a malesuada.

Inhaltsverzeichnis

Kapitel 1

Einführung

This statement requires citation [1]; this one does too [**?**]. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean dictum lacus sem, ut varius ante dignissim ac. Sed a mi quis lectus feugiat aliquam. Nunc sed vulputate velit. Sed commodo metus vel felis semper, quis rutrum odio vulputate. Donec a elit porttitor, facilisis nisl sit amet, dignissim arcu. Vivamus accumsan pellentesque nulla at euismod. Duis porta rutrum sem, eu facilisis mi varius sed. Suspendisse potenti. Mauris rhoncus neque nisi, ut laoreet augue pretium luctus. Vestibulum sit amet luctus sem, luctus ultrices leo. Aenean vitae sem leo.

Nullam semper quam at ante convallis posuere. Ut faucibus tellus ac massa luctus consectetur. Nulla pellentesque tortor et aliquam vehicula. Maecenas imperdiet euismod enim ut pharetra. Suspendisse pulvinar sapien vitae placerat pellentesque. Nulla facilisi. Aenean vitae nunc venenatis, vehicula neque in, congue ligula.

R

R ist eine Programmiersprache und Entwicklungsumgebung, die für statistische Berechnungen und Graphen unter John Chambers von Bell Laboratories entwickelt wurde. Sie ist ein GNU-Projekt, bei dem die Entwicklung von freier Software im Mittelpunkt steht. R weist große Ähnlichkeiten zu der Programmiersprache S auf, ein weiteres GNU-Projekt, welches weitgehend auch unter R läuft. [2]

R bietet standardmäßig alle Hauptfunktionen an für die statistische Analyse von Datensätzen an und ist einfach zu erweitern, weswegen es vor allem für wissenschaftliche Arbeiten verwendet wird. Es ist mit R einfach,

statistische Funktionen auf große Datenmengen anzuwenden.

ARX

ARX ist eine freie Software zur Anonymisierung von medizinischen Datensätzen, die von Fabian Prasser und Florian Kohlmayer vom Institut für medizinische Statistik und Epidemiologie an der Technischen Universität München entwickelt wurde.

Kapitel 2

Anwenderdokumentation

2.1 Übersicht

Das R-Terminal dient dazu, über eine externe Schnittstelle R aufzurufen und zu bedienen. Dies soll vor allem dazu genutzt werden, um Tabellen aus ARX einzuladen und Skripte auszuführen, und gleichzeitig die im Kapitel 1 beschriebenen Probleme zu umgehen. Das R-Terminal ist kompatibel mit Windows, der Linux Distribution Ubuntu und OS X, von denen jeweils die folgenden Versionen im Rahmen der Entwicklung getestet wurden:

- Windows 10 Education (Version 1511)
- OS X El Capitan (Version 10.11.1)
- macOS Sierra (Version 10.12.2)
- Ubuntu

In den folgenden Abschnitten werden die Grundfunktionen und zusätzliche Features des R-Terminals beschrieben.

2.2 Vorbereitungen

TODO Installation von R im Vorhinein und Auswahl der richtigen SWT-Libraries beschreiben. (Auswahl der Libraries wird in 3.5.1 beschrieben ;)

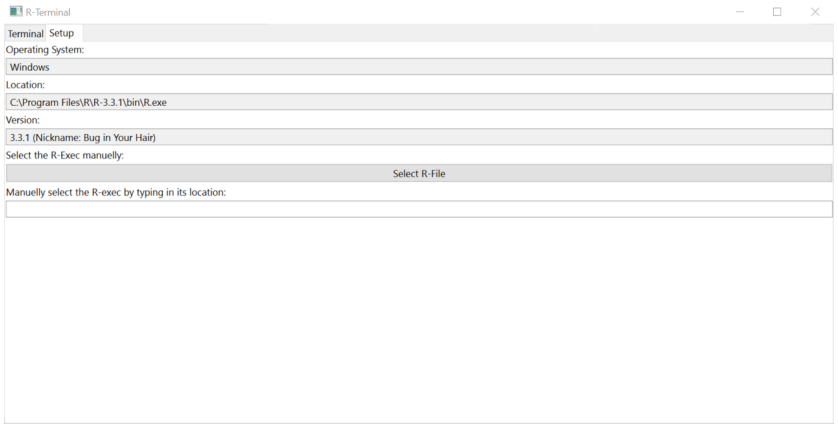

Abbildung 2.1: R-Terminal: *Terminal* unter Windows 10 Education (Version 1511)

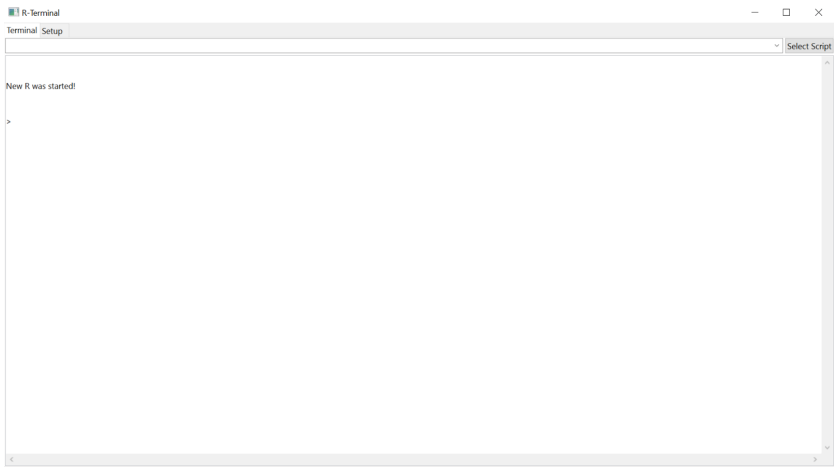

Abbildung 2.2: R-Terminal: *Setup* unter Windows 10 Education (Version 1511)

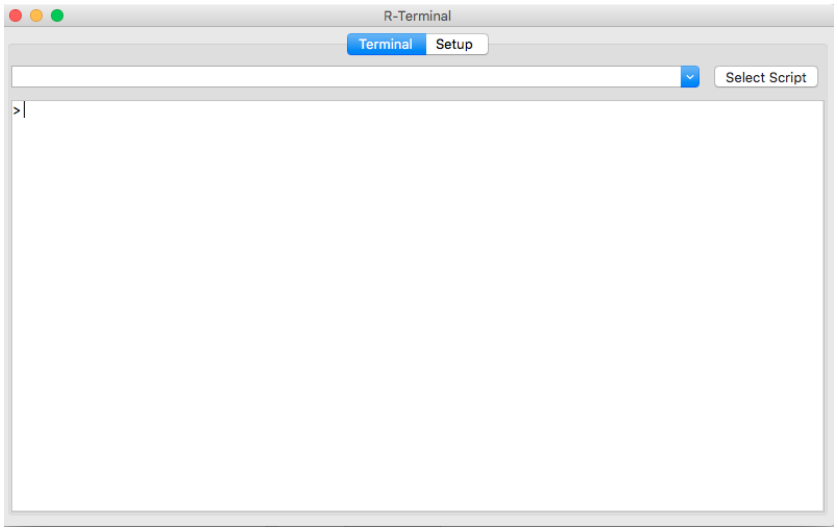

Abbildung 2.3: R-Terminal: *Terminal* unter OS X (Version 10.11.1)

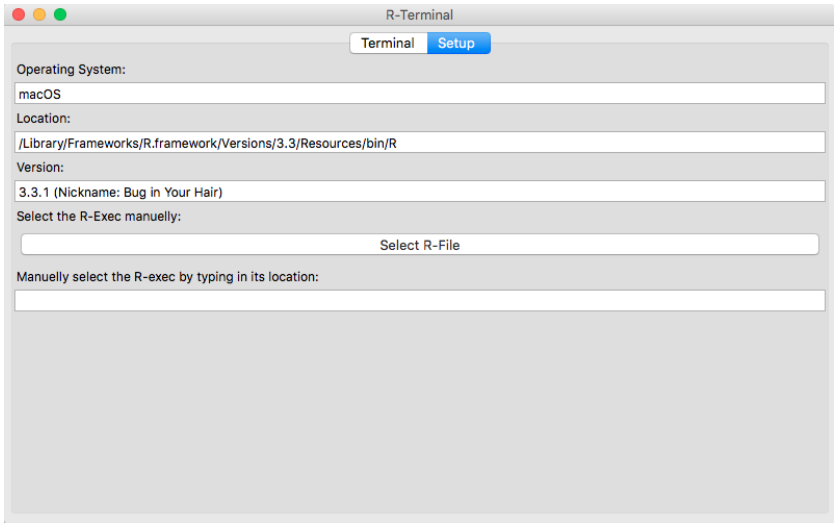

Abbildung 2.4: R-Terminal: *Setup* unter OS X (Version 10.11.1)

2.3

In Abbildung 2.3 ist das R-Terminal unter OS X (hier Version 10.11.1) zu sehen. Das Terminal verfügt über die beiden Tabs *Terminal* und *Setup* (s. **??**.

2.3.1 Setup

Unter *Setup* wird entweder eine R-Version auf dem Rechner gesucht und automatisch ausgeführt oder es wird vom Benutzer selber der Pfad zu der gewünschten R-Version angegeben.

Hierbei gibt das Feld unter *Operating System* das Betriebssystem der benutzten Plattform an. Das Feld *Location* gibt den Pfad an, unter dem die automatische Suche die ausführbare R-Version gefunden hat. Konnte keine ausführbare R-Version gefunden werden, wird in dem Feld die Nachricht *No valid R-exec found!* angezeigt.

Das mit *Version* betitelte Feld gibt die Version an, die die gefundene R-Version hat. Wurde keine ausführbare R-Version gefunden oder eine ungültige R-Version angegeben, zeigt das Feld die Ausgabe *No valid R-Version selected*an.

Sind auf dem Rechner mehrere R-Versionen vorhanden, von denen der Benutzer eine andere Version bevorzugt, als die durch die manuelle Suche gefundene, gibt es die beiden unteren, im folgenden erläuterten Felder als Optionen.

Select the R-Exec manually öffnet eine Ordnerübersicht, durch die dann die zu der gewünschten R-Version navigiert werden kann. Hierbei sollte beachtet werden, dass nicht der Pfad zu der R-GUI angegeben werden sollte, welche oft zusammen mit dem R-Executable installiert wird, sondern zu der ausführbaren R-Datei.

Das letzte Feld *Manually select the R-exec by typing in its location* gibt dem Benutzer die Möglichkeit, den bereits bekannten Dateipfad zur ausführbaren R-Datei anzugeben. Hierbei sollte ein absoluter Dateipfad angegeben werden, der die ausführbare R-Datei enthält (im Gegensatz dazu, nur den Ordner anzugeben, in dem die Datei liegt).

Wird in *Select the R-Exec manually* oder in *Manually select the R-exec by typing in its location* eine ungültige R Version oder ein ungültiger Dateipfad angegeben, wird unter *Operating System* und *Version* dieselbe Fehlermeldung angegeben, wie oben bei dem Fehlschlagen der automatischen Suche beschrieben.

2.3.2 Terminal

Nachdem die gewünschte R-Version gefunden wurde, kann diese nun unter dem Tab *Terminal* bedient werden. Das Eingabefeld kann benutzt werden, um R-kompatible Befehle auszuführen. Es werden bis zu 10 Befehle gespeichert. Diese können ausgewählt werden, indem auf den Pfeil auf der rechten Seite des Eingabefeldes geklickt wird. Dadurch öffnet sich ein Dropdown-Menü, aus dem der gewünschte Befehl ausgewählt werden kann. Dieser erscheint im Eingabefeld.

2.4 Benutzung von R für ARX

Dieser Abschnitt wird sich mit der Benutzung von R im Zusammenhang mit großen Datentabellen befassen, wie sie in ARX üblich sind. Dies soll die Benutzung von dem R-Terminal zur Verarbeitung von ARX-Daten ermöglichen.

2.4.1 Skalenniveau

In der medizinischen Statistik ist die Unterscheidung von den verschiedenen Stufen der Skalierbarkeit von großer Bedeutung. Für die Daten aus ARX sind vor allem die Eigenschaften wichtig, die Patientenmerkmale besitzen. So können Merkmale in quantitative und qualitative Merkmale unterteilt werden [3]. Es ist kein Problem, quantitative Merkmale darzustellen, da diese einfach durch numerische Werte ausgedrückt werden können. Die Grunddatentypen sind wie folgt:

- Numeric
- Integer
- Complex
- Logic
- Character

Wenn also eine Variable als Dezimalzahl ohne zusätzliche Bedingung deklariert wird, ist der Datentyp standardmäßig `numeric` (vgl. 2.4.1).

Listing 2.1: Deklarierung einer numerischen Variable x

```
x = 13.4
```

Will man explizit einen Integer als Variable initialisieren, sieht der Befehl aus wie in **??**.

Listing 2.2: Deklarierung einer numerischen Variable x

```
x = 13.4
```

Kapitel 3

Entwicklerdokumentation

3.1 Verwendete Technologien

Das *R-Terminal* wurde mit der Entwicklungsumgebung *Eclipse* entwickelt. Als Programmiersprache wurde *Java* gewählt. Die graphische Benutzeroberfläche wurde mit *SWT* realisiert, genauere Details hierzu werden in 3.5.1 erläutert. Um *R*-Kommandos auszuführen, startet das hier dokumentierte Programm *R-Terminal* einen *R*-Prozess der Standalone-Installation von *R* durch die Java-Library *ProcessBuilder*.

3.2 Architektur des R-Terminals

Die Software des *R-Terminal* ist in zwei Komponenten aufgeteilt, welche als getrennte Packages vorliegen. Das erste Package „org.deidentifier.arx.gui“ beinhaltet die graphische Benutzeroberfläche, das Zweite „org.deidentifier.arx.r“ beinhaltet die gesamte Integration von *R* in *Java*, es stellt also alle funktionalen Methoden zur Verfügung und verwaltet die Kommunikation zwischen *R* und dem *Java*-Programm.

Durch die Trennung von graphischer Benutzeroberfläche und der Integration von *R*, also dem funktionalen Kern, kann die *R*-Integration auch ohne graphische Benutzeroberfläche ausgeführt und verwendet werden. Hierzu wird nur das zweite Package benötigt, welches in 3.4 behandelt wird. Die Ausführung ohne GUI wird in 3.6 erläutert.

Alle Klassen der Software *R-Terminal*, sowie deren wichtigsten Methoden, werden in den Kapiteln 3.3 und 3.4 vorgestellt.

3.3 Graphische Benutzeroberfläche (GUI)

Das erste Package, „org.deidentifier.arx.gui“, erzeugt und verwaltet die graphische Benutzeroberfläche. Es umfasst sieben Klassen:

- RMain
- RTerminal
- RSetupTab
- RTerminalTab
- RBrowserWindow
- RLayout
- RCommandListener

3.3.1 RMain

Um das Programm mit einer graphischen Benutzeroberfläche zu starten, muss die „main“-Methode der Klasse *RMain* ausgeführt werden.

Diese erzeugt ein neues Display sowie eine Shell. Im Anschluss wird ein neues Objekt der Klasse *RTerminal* erzeugt.

3.3.2 RTerminal

Die Erzeugung der einzelnen Komponenten der graphischen Benutzeroberfläche wird durch den Konstruktor der Klasse *RTerminal* realisiert.

Dieser erzeugt einen TabFolder mit zwei Tabs, der erste Tab beinhaltet ein Objekt der Klasse *RTerminalTab*, der zweite ein Objekt der Klasse *RSetupTab*. Außerdem erzeugt der Konstruktor noch einen Ring-Puffer „buffer“, welcher zur Speicherung des Std-Output von *R* verwendet wird, sowie einen Listener „listener“. Sowohl der Ring-Puffer, als auch der Listener wurden im Package „org.deidentifier.arx.r“ implementiert. Im Anschluss wird die Integration mit *R* durch den Aufruf der Methode „startRIntergration“ eingeleitet.

Diese Methode „startRIntergration“ erzeugt ein neues Objekt der Klasse „RIntegration“, welches in 3.4.1 beschrieben wird und die Integration von *R* realisiert.

TODO Listener Methodenüberschreiben

3.3.3 RSetupTab

Die Klasse *RSetupTab* erzeugt einen Tab, welcher Informationen zum verwendeten Betriebssystem sowie dem aktuellen Status der Integration von *R* anzeigt.

Die unterschiedlichen Informationen werden in *SWT*-Labels dargestellt, welche in einem *GridLayout* angeordnet sind. Die *SWT*-Labels werden durch die Methodenaufrufe „showOS()“, „showRLocation()“, „showRVersion()“ im Konstruktor erzeugt.

Um das verwendete Betriebssystem abzufragen, wird die Methode „printOS()“ der Klasse *OS*, siehe 3.4.2, von „showOS()“ aufgerufen. Diese gibt das verwendete Betriebssystem als String zurück.

Die Integration von *R* wird in der Klasse *RIntegration* des Package „org.deidentifier.arx.r“ umgesetzt. Diese wird beim Start des Programms in *RTerminal* aufgerufen. Um den Status der Integration zu prüfen, rufen „showRLocation()“ und „showRVersion()“ die Methode „getR()“ aus *OS* auf. Diese gibt den absoluten Pfad zur aktuell verwendeten R-Executive zurück, welcher im *RSetupTab* angezeigt wird. Wurde keine R-Executive gefunden oder konnte diese nicht erfolgreich gestartet werden, so wird von „getR()“ *null* zurückgegeben und im *RSetupTab* wird ausgegeben:

Location: „No falid R-exec found!“
Version: „No falid R-Version selected!“

Wurde eine *R*-Version gefunden und erfolgreich ausgeführt, so wird der erzeugte Listener in *RTerminal* ausgelöst. Dieser ruft anschließend die Methode „update()“ auf, durch welche der Inhalt von Location und Version aktualisiert wird.

Außerdem ermöglicht der *RSetupTab* die manuelle Auswahl einer *R*-Executive. Im Konstruktor wird hierfür ein Knopf mit „createManuellSearchWindow()“ zum Öffnen eines Navigationsfensters sowie eine Kommandozeile „createDirSearchLine()“ erzeugt, sodass entweder der absolute Pfad zur Datei angegeben oder diese mittels des Navigationsfensters ausgewählt werden kann.

Die Kommandoleiste wird in der Methode „createDirSearchLine()“ erzeugt und erfasst die Eingabe durch einen *TraverseListener*. Der *TraverseListener* wird durch Drücken der Return-Taste ausgelöst und gibt den eingegeben Pfad an die Methode „updateSetup(*String* path)“ weiter. Wird

die *R*-Executive mit dem Navigationsfenster gesucht, so wird der absolute Pfad der ausgewählten Datei ebenfalls an „updateSetup(*String* path)“ als Argument übergeben. Nähere Informationen zum Navigationsfenster finden sich in 3.3.5.

Die Methode „updateSetup(*String* path)“ startet eine neue Integration von *R* durch den Aufruf der Methode „startManuellRIntegration(*String* path)“ aus der Klasse *RTerminal*. Nähere Details hierzu in 3.3.4.

Falls die *R*-Executive erfolgreich gestartet wurde, wird durch den *RListener* (siehe oben) der Tab wieder aktualisiert. Andernfalls wird die selbe Ausgabe angezeigt, wie bei der automatischen Suche.

3.3.4 RTerminalTab

Im *RTerminalTab* werden *R*-Befehls-Eingaben des Nutzers erfasst sowie der Std-Output von *R* ausgegeben. Außerdem können mittels eines Knopfes fertige *R*-Skripte geladen und ausgeführt werden. Der *RTerminalTab* ist nur nutzbar, falls *R* erfolgreich gestartet und integriert wurde.

Der Tab hat als Layout ebenso wie der *RSetupTab* ein *GridLayout*. Dieses besteht aus einem *Composite* und einem Textfeld.

Das Kompositum „topline“ umfasst eine Kommandozeile, ein Dropdown-Menü um eingegebene Befehle erneut auszuführen sowie einen Knopf zum Aufrufen von *R*-Skripten. Als Layout wurde ebenfalls *GridLayout* gewählt, da es alle Komponenten möglichst kompakt darstellt.

Die Kommandozeile und das Dropdown-Menü „input“ wurden mit der importierten Klasse *swt.widget.Combo* erstellt. Die Kommandozeile erfasst die Eingaben des Nutzers beim Drücken der Enter-Taste durch einen *TraverseListener*. Die Eingabe wird anschließend an die Methode „command(*String* command)“ der Klasse *RCommandListener* übergeben, welche den Befehl an *R* übergibt. Dies wird in 3.3.7 genauer beschrieben.

Die letzten 10 eingegebenen Befehle werden im Dropdown-Menü angezeigt. Die Befehle werden in einem Ring-Puffer mit 10 Elementen gespeichert, welcher als String Array implementiert wurde.

Skripte können durch den Knopf „scriptButton“ ausgewählt und ausgeführt werden. Der Knopf wird durch einen *MouseListener* ausgelöst und erzeugt ein Navigationsfensters der Klasse *RBrowserWindow*. Weiter Informationen hierzu in 3.3.5. Durch dieses kann das Skript ausgewählt werden

und es wird vom *RBrowserWindow* der absolute Pfad des Skriptes übergeben. Anschließend wird überprüft, ob es sich um ein *R*-Skript handelt, also die Datei auf „.r“ endet. Trifft dies zu, so wird durch die Methode „command(*String* command)“ der Klasse *RCommandListener* der Befehl das Skript zu öffnen an *R* übergeben. Dieser setzt sich zusammen aus:

$$source(\text{" absoluter Pfad "}).$$

Das Textfeld „output“ beinhaltet die Ausgabe von *R* und wurde durch die importierte Klasse *StyledText* realisiert. Das Textfeld ist als Ring-Puffer implementiert, sodass dieser nur die letzten 10000 Zeichen des *R*-Std-Output anzeigt. Die Größe des Ring-Puffers ist in der Klasse *RTerminal* als globale int Variable „BUFFER_SIZE“ festgelegt und kann hier geändert werden. Der Std-Output von *R* wird durch das Auslösen des RListener „listener“ in *RTerminal* aktualisiert. Dieser ruft hierzu die Methode „setOutput(*String* text)“ aus *RTerminalTab* auf. Der Listener wird nach erfolgreichem Starten von *R* durch die Klasse *RIntegration* übergeben. Diese ruft die Methode „setCommandListener(*RCommandListener* listener)“ im *RTerminalTab* auf und übergibt den Listener als Argument. Details zum Listener finden sich in 3.3.7.

Außerdem wird nach einem erfolgreich Start von *R* durch „startRIntegration(*String* path)“ aus der Klasse *RTerminal* die Methode „enableTab()“ in *RTerminalTab* aufgerufen, sodass der Tab nutzbar wird. Beim Beenden von *R* wird in der Methode „endR()“ in *RTerminal* die Methode „disableTab()“ in *RTerminalTab* aufgerufen, welche den Tab wieder für den Nutzer sperrt.

3.3.5 RBrowserWindow

Die Klasse *RBrowserWindow* implementiert ein Navigationsfenster des Standart-Dateimanagers. Dieses wurde durch die importierte *SWT*-Klasse *FileDialog* implementiert. Die Klasse beinhaltet nur eine Methode „openBrowser(Shell)“, durch welche ein neuer *FileDialog*, also ein Navigationsfenster, erzeugt wird, mit welchem eine Datei ausgewählt werden kann. Anschließend gibt die Methode den absoluten Pfad der ausgewählten Datei zurück. Mit dem Navigationsfenster kann sowohl manuell die *R*-Executive im *RSetupTab* sowie ein ausführbares *R*-Skript im *RTerminalTab* ausgewählt werden. Anschließend wird der absolute Pfad per *return* übergeben, sodass der funktionale Kern, siehe 3.4, die jeweilige Aktion durchführen kann.

3.3.6 RLayout

Diese Klasse definiert das *SWT*-Layout der Klassen *RSetupTab* und *RTerminalTab*. Die Methoden der Klasse werden im Konstruktur der beiden genannten Klassen aufgerufen, um die *SWT*-Objekte der des jeweiligen Tabs mit den richtigen Parametern und Abständen zu erzeugen.

3.3.7 RCommandListener

Das Java-Interface *RCommandListener* implementiert einen Listener, welcher bei der Eingabe von Befehlen für *R* ausgeführt wird. Die abstrakte Methode „command(*String* command)“ des Interfaces wird in der Klasse *RTerminal* mit der Methode „execute(command)“ aus der Klasse *RIntegration* überschrieben. Somit wird die Trennung der graphischen Oberfläche und des funktionalen Kerns der *R*-Integration, welcher in 3.4 beschrieben wird, beibehalten.

3.4 Funktionaler Kern - Integration von R

Die Integration von *R* in Java ist in dem Package „org.deidentifier.arx.r“ implementiert. Dieses beinhaltet alle Funktionen, durch welche die *R*-Executive automatisch gefunden, ausgeführt und das Betriebssystem erkannt wird. Außerdem beinhaltet es alle Methoden zur Übermittlung des Input- und Outputstreams. Zur Integration von *R* in Java wurde die Klasse *ProcessBuilder* verwendet.

Das Package umfasst vier Klassen:

- RIntegration
- OS
- RBuffer
- RListener

3.4.1 RIntegration

Die Klasse *RIntegration* ist das Kernstück des *R-Terminals* und ist für die Integration von *R* verantwortlich. Der *R*-Prozess wird durch die Library *ProcessBuilder* gestartet.

Wird das Programm mit der graphischen Benutzeroberfläche gestartet, so wird für jede gefundene ausführbare *R*-Executive durch die Methode „startRIntegration“ beziehungsweise „startManuellRIntegration“ ein neues Objekt dieser Klasse erzeugt. Hierbei werden dem Konstruktor folgende Argumente übergeben:

- *final String* path
- *final buffer* buffer
- *final RListener* listener

Der absolute Pfad zur Executive wird durch „path“ angegeben. Bei „buffer“ handelt es sich um den Ring-Puffer der Klasse *RBuffer*, welcher zur Ausgabe des Std-Output des *R*-Prozesses im *RTerminalTab* erzeugt wurde. Gibt es eine neue Ausgabe des Std-Output des Prozesses, so wird dieser im Ring-Puffer angehängt. Um die beiden Tabs *RSetupTab* und *RTerminalTab* bei Änderungen zu aktualisieren, wird der *RListener*, welcher in *RTerminal* erzeugt wurde, als Attribut „listener“ übergeben. Bei einer Ausgabe des Std-Output oder Start bzw. Beenden des *R*-Prozesses wird dieser ausgelöst.

Der Konstruktor erzeugt ein neues Objekt der Klasse *ProzessBuilder* mit den betriebssystemspezifischen Parametern. Die Parameter werden durch Aufrufen der Methode „getParameters(*String* path)“, siehe 3.4.2, aus der Klasse *OS* übergeben und dem Konstruktor des *ProzessBuilders* als Argumente übergeben.
Anschließend wird der Prozess gestartet.

Nach erfolgreichen Start des *R*-Prozesses wird ein Reader „reader“ erzeugt, welchem die Ausgabe des *R*-Prozesses übergeben wird. Dieser Reader übergibt die Ausgabe an den Ring-Puffer „listener“.
Direkt im Anschluss wird die Methode „getVersion(*Reader* reader)“ aufgerufen. Durch diese wird abgefragt, welche Version von *R* verwendet wird. Die Ausgabe der Version wird durch das Kommando „version“, welches an den laufenden *R*-Prozess als Input durch den Methodenaufruf „execute("version")“ übergeben wird, erreicht. Die Ausgabe dieses Kommandos soll nicht im Ausgabefenster der GUI, dem *RTerminalTab*, angezeigt werden. Deshalb erzeugt die Methode „getVersion(Reader reader)“ einen neuen, temporären Ring-Puffer, in welchem die Ausgabe für diesen Befehl gespeichert wird.

Aus dem Ausgabe-String dieses „version“-Kommandos wird anschließend die genaue Version sowie der Nickname extrahiert und in der lokalen Variable „version“ gespeichert. Im Anschluss wird der Rlistener „listener“ ausgelöst, welcher die Methode „setupUpdate()“ ausführt. Diese Methode ruft die „update“-Methode im *RSetupTab* auf und aktualisiert den Tab.

Nach Abschluss dieses Kommandos wird die gesamte Ausgabe in den Ring-Puffer „buffer“ geschrieben und durch Auslösen des RListener „listener“ in dem Ausgabefenster des *RTerminalTabs* angezeigt.

Kommandos, welche in *R* ausgeführt werden sollen, werden als String der Methode „execute(*String* command) “ übergeben, welche diese an den *R*-Prozess weiterleitet. Hier wird ein *BufferedWriter*-Objekt erstellt. Mit der „write(String command)“ Methode dieses Elementes wird das Kommando in dem *BufferedWriter* geschrieben und anschließend mit „newline()“ noch ein Zeilenumbruch im *BufferedWriter* gespeichert. Um den Inhalt des *BufferedWriter* an den *R*-Prozess zu übergeben, diesen auszuführen und anschließend den *BufferedWriter* zu leeren, wird zum Abschluss die Methode „flush()“ aufgerufen. Mit dieser Methode werden alle Kommandos an den *R*-Prozess übergeben. Die „execute(*String* command)“ Methode wird bei der Nutzung der GUI von der Methode „command(*String* command)“ aufgerufen. Nähere Informationen zur Methode „command(*String* command)“ finden sich in 3.3.7.

3.4.2 OS

Die Klasse *OS* implementiert folgende Funktionalitäten: Erkennung des verwendeten Betriebsystems, die Anpassung der Pfade der Standart-Speicherorte der *R*-Executive je nach verwendeten Betriebssystem und die Überprüfung der übergebenen *R*-Executives.

Um die drei unterstützten Betriebssysteme Windows, Mac und Unix effizient zu unterscheiden, werden diese als Enum Konstanten „OSType“ erzeugt.

Die absoluten Pfade der Standart-Speicherorte der *R*-Executive werden für jedes Betriebssystem in einem eigenen String Array gespeichert. Es ist wichtig hier zu beachten, dass Windows andere Separatoren verwendet als Linux und OS X. Bei Windows müssen die einzelnen Verzeichnisse durch

„\\“ getrennt werden, bei beiden anderen Betriebssystemen durch „/“.

Falls neue Standart-Speicherorte für die *R*-Executive hinzugefügt werden sollen, kann das jeweilige Array einfach um den neuen Pfad erweitert werden. Beim Starten des *R-Terminals* durch die graphische Benutzeroberfläche werden alle Pfade des Arrays nach einer ausführbaren *R*-Executive durchsucht.Hierfür wird die Methode „getR()“ aufgerufen, welche im Folgenden noch vorgestellt wird.

Außerdem werden die Dateinamen der *R*-Executive für die unterschiedlichen Betriebssysteme in einem String-Array gespeichert, da sich diese je nach Betriebssystem unterscheiden. Außerdem wird das jeweilige Array benötigt, um bei der manuellen *R*-Auswahl zu prüfen, ob es sich um eine gültige *R*-Executive handelt. Dies wird in der Methode „isRExec(*String* path)“ in Abhängigkeit vom verwendeten Betriebssystems überprüft.

Die Methode „getOS()“ fragt den Namen des Betriebssystems ab und gibt dieses als Enum „OSType“ passend zurück.

Wie bereits beschrieben, werden nach dem Start des Programms die Standart-Speicherorte nach einer ausführbaren R-Executive durchsucht. Ob an einem dieser Speicherorte eine Executive liegt, wird mit der Methode „getPath(*String*[] locations, *String*[] executables)“ überprüft. Außerdem wird versucht die Executive mit *ProcessBuilder* zu starten, um zu prüfen, ob der Nutzer ausreichende Zugriffsberechtigungen besitzt.

Zusätzlich sind in der Klasse *OS* die Parameter zur Ausführung der *R*-Executive für den *ProcessBuilder* gespeichert. Diese werden beim Aufruf der Methode „getParameters(*String* path)“ mit dem Pfad „path“ der Executive zu einem String-Array konkateniert und anschließend als String-Array zurückgegeben.

3.4.3 RBuffer

Diese Klasse implementiert den Ring-Puffer, welcher für die Ausgabe des Std-Output von *R* im Ausgabefenster des *RTerminalTab* verwendet wird. Die Größe des Ring-Puffers wird bei der Erzeugung dem Konstruktor übergeben und ist beim Starten des Programms mit graphischer Benutzeroberfläche in der Klasse *RTerminal* in der Variable „BUFFER_SIZE“ festgelegt.

3.4.4 RListener

Die Klasse *RListener* implementiert einen Listener, welcher in den Klassen *RTerminal*, siehe 3.3, sowie *RIntegration* verwendet wird.

Der Listener aktualisiert die Inhalte von *RSetupTab* sowie *RTerminalTab*. Durch die abstrakte Methode „setupUpdate()“, welche in *RTerminal* überschrieben wird, wird bei Änderung des Status von *R*, also dem Beenden oder dem Starten eines neuen *R*-Prozesses der *RSetupTab* aktualisiert. Durch „bufferUpdate()“ wird das Ausgabefenster im *RTerminalTab* aktualisiert. Diese abstrakte Methode wird ebenfalls in *RTerminal* überschrieben.

3.5 Abhängigkeiten

Das Projekt beinhaltet bereits alle benötigten Java-Libraries zur Ausführung der Software. Da die graphische Benutzeroberfläche mit *SWT* realisiert wurde, müssen die *SWT*-Libraries je nach Betriebssystem ausgewählt werden. Dies wird genauer unter 3.5.1 erläutert.

Für die Integration von *R* in Java, welche durch das *R-Terminal* umgesetzt wurde, muss eine Standalone-Installation von *R* vorliegen. Nähre Details hierzu in 3.5.2.

3.5.1 SWT

Die graphische Benutzeroberfläche wurde mithilfe des *Standard Widget Toolkit* (*SWT*) realisiert. *SWT* ist ein „open-source widget toolkit“ für Java mit Integration in die GUI des nativen Betriebssystems, sodass das Design der Benutzeroberfläche an das Design des Betriebssystems angepasst wird.
Für das Projekt *R-Terminal* wurde die *SWT*-Version 4.2.1 verwendet.

Wie die betriebssystemspezifischen *SWT*-Libraries auf den „Build Path“ hinzugefügt werden, kann unter 3.5.1 nachgeschlagen werden.

TODO vlt in die Anwenderdokumentation?

Um das *R-Terminal* ausführen zu können, müssen die betriebssystemspezifischen *SWT*-Libraries auf den Bild-Path gesetzt und die *SWT*-Libraries

anderer Betriebssysteme entfernt werden. Im Folgenden wird genau beschrieben, wie dies unter *Eclipse* umgesetzt wird:

1. Navigieren Sie im *Eclipse* Package Explorer zu:

   R-Terminal -> lib -> swt

2. Wählen Sie die betriebssystemspezifischen *SWT*-Libraries aus und öffnen Sie mit der rechten Maustaste, bzw. dem OS-spezifischen Äquivalent, für jede dieser Libraries das Auswahlfenster. Klicken Sie auf „Build-Path“. Sollte die Library bereits hinzugefügt sein, erscheint „Configure Build Path...“. Ist dies nicht der Fall, erscheint „Add to Build Path“, klicken Sie dieses an.

   Betriebssystemspezifische *SWT*-Libraries:

   - Windows:
     - swt-4.2.1-win32-win32-x86_64.jar
     - swt-4.2.1-win32-win32-x86.jar
   - Mac:
     - swt-4.2.1-cocoa-macosx-x86_64.jar
     - swt-4.2.1-cocoa-macosx.jar
   - Linux:
     - swt-4.2.1-gtk-linux-x86_64.jar
     - swt-4.2.1-gtk-linux-x86.jar

3. Überprüfen Sie nun, ob alle anderen Libraries des Ordners *swt* nicht auf dem „Build Path“ liegen. Gehen Sie hierfür zu:

   R-Terminal -> Referenced Libraries

   Überprüfen Sie, dass keine der genannten Libraries der oberen Aufzählung, die nicht Ihrem Betriebssystem zugehört, in diesem Ordner enthalten ist. Sollte sich eine *SWT*-Library für ein anderes Betriebssystems in diesem Ordner befinden, öffnen Sie mit einem Rechtsklick das Menü und wählen Sie aus:

   Build Path -> Remove from Build Path

Nun können Sie das Java-Programm starten.

3.5.2 R-Project

Um *R* in Java zu integrieren muss eine Standalone-Installation von R, für welche der aktuellen Benutzer ausreichende Zugriffsrechte besitzt, vorhanden sein. Alle aktuellen *R*-Versionen können von der Homepage des *R*-Projects[1] heruntergeladen werden.

3.6 Ausführung ohne graphische Benutzeroberfläche

Der funktionale Kern des *R-Terminals* kann auch ohne die graphische Benutzeroberfläche ausgeführt werden. Hierfür müssen allerdings folgende Objekte erzeugt werden:

- ein Ring-Puffer der Klasse *RBuffer*, welcher den Input für *R* beinhaltet
- ein weiterer Ring-Puffer der selben Klasse, welchen den Std-Output von *R* übergeben wird
- ein Listener der Klasse *RListener*, welcher durch den Std-Output ausgelöst wird

Anschließend müssen die abstrakten Methoden des *RListeners* überschrieben werden:

- *bufferUpdated()*
- *closed()*
- *setupUpdate()*

Anschließend kann die Integration von *R* gestartet werden. Hierfür muss ein neues *RIntegration*-Objekt erzeugt werden.

Um dem *R*-Prozess nun Befehle zu übergeben, muss lediglich die Methode „execute“ des erzeugten *RIntegration*-Objekt aufgerufen werden und das Kommando als String übergeben werden.

Beispiel: Es wurde ein *RIntegration*-Objekt *rProcess* erzeugt und es soll „1+2+3+4+5“ als Kommando übergeben werden, dann muss

rProcess.execute(„1+2+3+4+5“)

[1] `https://www.r-project.org`

ausgeführt werden. Die Ausgabe wird im Ring-Puffer gespeichert, welcher für den Std-Output erzeugt wurde.

Ein Code-Beispiel hierzu befindet sich im Package „ExampleWithoutGUI“.

Kapitel 4

Designentscheidungen

TODO warum dieses Design

In diesem Kapitel werden mögliche Alternativen vorgestellt um *R* in ein *Java*-Programm zu integrieren. Die in dieser Dokumentation vorgestellte Umsetzung wurde gewählt, um die verwendete Version von *R* frei wählen zu können und des weiteren Abhängigkeiten von weiteren Libraries zu vermeiden.

4.1 Alternativen

4.1.1 rJava

TODO

4.1.2 JRI

TODO

Literaturverzeichnis

1. Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1993.

2. `https://www.r-project.org/about.html`

3. `https://biometrie.charite.de/fileadmin/user_upload/microsites/m_cc04/biometrie`