Smarte Integration von R im SAS Data Step

Hardy Armes
Condat AG
Alt-Moabit 91d
10559 Berlin
Hardy.Armes@condat.de

Zusammenfassung

Trotz einer Vielzahl von Schnittstellen des SAS Systems zu anderen Systemen, ist es bisweilen nützlich, eine eigene Implementierung einer Schnittstelle zu haben. Die Gründe hierfür könnten verschiedener Natur sein. Beispielsweise der Wunsch nach bestimmter Funktionalität, die von der SAS-Standardschnittstelle nicht geboten wird, oder bestimmten erforderlichen Datenkonvertierungen oder wegen nicht erfüllbarer Voraussetzungen, wie eine bestimmte geforderte Version des anderen Systems, welche nicht verwendet werden kann, oder schlicht weil es keine Schnittstelle gibt. Im Zusammenspiel einiger interessanter „Zutaten" ist es im SAS Data Step möglich, eigene Schnittstellen, insbesondere zu Open Source Systemen, zu implementieren. Ein Beispiel hierfür soll die Implementierung einer Schnittstelle zu R sein, welche in diesem Beitrag vorgestellt wird.

Schlüsselwörter: FCMP-Prozedur, SAS Makro, SAS Data Step, Java Object, Java, R

1 Worum es geht

SAS bietet bereits in der Basissoftware (Base SAS) viele nützliche und interessante Werkzeuge, um die Kommunikation mit anderen Systemen zu unterstützen. Im Hinblick auf Open Source Software ist dabei insbesondere das Java Object vom Data Step Component Interface von Interesse, mit dem ermöglicht wird, auf Basis von Java-Technologie weitere Funktionalität im SAS System verfügbar zu machen[1]. Obwohl es eine Reihe von SAS Produkten gibt, mit denen ebenfalls die Kommunikation mit anderen Systemen unterstützt wird (wie beispielsweise die vielzähligen ACCESS-Produkte), so ist die Möglichkeit einer eigenimplementierten Schnittstelle dennoch von Interesse, zum einen weil Schnittstellen zu Open Source Software eher eine Ausnahme sind, zum anderen aber weil die Schnittstelle um eigene Aspekte ergänzt werden kann (wie beispielsweise spezielle Datenkonvertierungen), die von einer ggf. bereits existierenden Schnittstelle nicht abgedeckt werden. Außerdem sind keine zusätzlichen SAS Produkte erforderlich. Obwohl es mittlerweile in SAS (seit der Version 9.22) ebenfalls eine Schnittstelle zu R gibt[2], ist eine eigene Schnittstelle aus den oben genannten Gründen dennoch interessant.

[1] Vgl. auch Poster „SAS DataStep Component Interface – Neue Objekte im DataStep" von S. Reimann in [1].
[2] Vgl. „Das Beste aus zwei Welten Aufruf von R-Funktionen mit PROC IML" von P. Bewerunge in [1].

2 Was R ist

R ist eine Software-Umgebung und Programmiersprache für Datenanalyse und Statistik[3]. R ist eine frei verfügbare Open Source Software, die von einer breiten Community mit neuen Methoden und Erweiterungen kontinuierlich ausgebaut wird. Entwicklungszyklen für neue Methoden verlaufen im Allgemeinen relativ schnell. R besitzt eine große Anzahl von bereits eingebauten Funktionen zur Verarbeitung von Matrizen und Vektoren und anderen Datenstrukturen und durch die leichte Erweiterbarkeit mittlerweile eine immense Bibliothek an implementierten Algorithmen für Datenzugriffe, Datenmanipulationen, Datenanalysen und grafische Auswertungen und Präsentationen. In jüngster Zeit erfreut sich R einer immer größer werdenden Anzahl von Nutzern, selbst unter SAS-Anwendern.

3 Schnittstellen zu R

Dieses Kapitel fasst die üblichen unterschiedlichen Wege von SAS auf R zuzugreifen zusammen[4], erklärt beispielhaft deren Arbeitsweise aus Sicht des Anwenders und nennt Vor- und Nachteile der jeweiligen Methode. Allen Verfahren gemein ist, dass mit ihnen Daten zwischen SAS und R ausgetauscht werden können, und dass die R-Funktionalität von SAS aus genutzt werden kann:

- Datentransfer zwischen SAS und R
- Aufruf von R-Funktionen innerhalb von SAS
- Transfer von Ergebnissen von R nach SAS

3.1 SAS/IML

Seit der SAS Version 9.22 stellt SAS ebenfalls eine Schnittstelle zu R mit dem Produkt SAS/IML bereit[5]. Mittels dieser Schnittstelle kann vom Anwender R-Code direkt an R zur Ausführung übergeben werden. Dies erfolgt mittels der IML-Prozedur innerhalb der Anweisungen SUBMIT / R und ENDSUBMIT. Die IML-Prozedur bietet die oben genannten grundsätzlichen Funktionen zum Datentransfer und zur Nutzung von R-Funktionalität.

Der Datenexport schließt SAS Datasets und IML-Matrizen (einer speziellen in SAS/IML verwendeten Datenstruktur, die es in analoger Form auch in R gibt) ein. Aus R können Dataframes und Matrizen importiert werden. Beim Datenexport von SAS Datasets werden neben numerischen und alphanumerischen Variablen, auch teilweise SAS-Formate für Datum (z.B. Date*w.d*), Zeit (z.B. Time*w.d*) und Zeitstempel (z.B. Datetime*w.d*) an R übertragen und korrespondierende R-Objekte angelegt. Beim Da-

[3] Vgl auch http://de.wikipedia.org/wiki/R_(Programmiersprache) bzw. http://www.r-project.org/.
[4] Der Vollständigkeit wegen, sollte nicht verschwiegen werden, dass es auch Möglichkeiten gibt seitens R auf SAS Datasets zuzugreifen.
[5] Vgl. auch http://www.sas.com/technologies/analytics/statistics/iml/index.html.

tenimport werden analog feste SAS-Formate für Datum- und Zeit-Objekte aus R zugewiesen. Allerdings können einige in R verwendete Datentypen (z.B. R-Objekte vom Typ Complex) und -strukturen (z.B. R-Objekte vom Typ List) nicht direkt in SAS importiert werden. Datenstrukturen müssen mittels R zuvor in einen Dataframe bzw. in eine Matrix umgewandelt werden. Text-Ergebnisse werden direkt im Output-Fenster und Logbuchausgaben im LOG-Fenster angezeigt. Grafische Ausgaben von R werden in speziellen Fenstern angezeigt oder können über externe Dateien weiterverarbeitet werden.

Beispiel

Das folgende Programmbeispiel zeigt die prinzipielle Anwendungsweise mittels IML-Prozedur anhand des SAS Datasets *Sashelp.Class* und der Ausführung einer einfachen linearen Regression zwischen den Variablen *Weight* und *Height*. Hierbei geht es nicht um den fachlichen Hintergrund des Beispiels, sondern vielmehr um die technischen Aspekte.

Zunächst werden die Daten von SAS nach R mit der ExportDataSetToR-Routine transportiert und daraus ein R-Dataframe namens *Class* erzeugt. Anschließend wird im SUBMIT-Block die Modellanpassung berechnet. Zwischen den Anweisungen SUBMIT / R und ENDSUBMIT steht R-Code, der vom R-Interpreter ausgeführt wird[6]. Damit die Ergebnisdaten mit SAS weiterverarbeitet werden können, müssen sie zum Schluss mittels R in einen Dataframe (oder in eine Matrix) transformiert werden, die sodann mit der ImportDataSetFromR-Routine in einen SAS Dataset importiert werden kann.

```
proc iml;
  run ExportDataSetToR("Sashelp.Class", "Class");
  submit / R;
    lmModel <- lm(Weight ~ Height, data=Class)
    Model <- data.frame(lmModel$model, Residuals=lmModel$residuals,
      Effects=lmModel$effects, FittedValues=lmModel$fitted.values)
  endsubmit;
  run ImportDataSetFromR("Work.Model", "Model");
quit;
proc print data=Model;
run;
```

Der erzeugte SAS Dataset *Work.Model* kann anschließend durch entsprechende SAS-Anweisungen weiterverarbeitet werden, hier im Beispiel durch die PRINT-Prozedur.

[6] Da die Programmiersprache R case-sensitiv ist, muss auf eine korrekte Schreibweise geachtet werden, beispielsweise um die betreffenden Variablen *Weight* und *Height* im Dataframe *Class* zu referenzieren.

The SAS System

Obs	Weight	Height	Residuals	Effects	Fitted Values
1	112.5	69.0	-13.5062	-436.005	126.006
2	84.0	56.5	6.7317	84.813	77.268
3	98.0	65.3	-13.5798	-10.508	111.580
4	102.5	62.8	0.6678	2.938	101.832
5	102.5	63.5	-2.0615	0.433	104.562
6	83.0	57.3	2.6125	3.118	80.388
7	84.5	59.8	-5.6351	-4.328	90.135
8	112.5	62.5	11.8375	14.011	100.662
9	84.0	62.5	-16.6625	-14.489	100.662
10	99.5	59.0	12.4841	13.535	87.016
11	50.5	51.3	-6.4933	-7.914	56.993
12	90.0	64.3	-17.6807	-14.929	107.681
13	77.0	56.3	0.5115	0.696	76.488
14	112.0	66.5	-4.2586	-0.801	116.259
15	150.0	72.0	12.2967	17.519	137.703
16	128.0	64.8	18.3698	21.281	109.630
17	133.0	67.0	14.7919	18.410	118.208
18	85.0	57.5	3.8327	4.402	81.167
19	112.0	66.5	-4.2586	-0.801	116.259

Abbildung 1: Ausgabe der PRINT-Prozedur

3.2 Systemkommando

Eine alternative Methode, R-Funktionalität innerhalb von SAS zu nutzen, besteht darin, R.exe mittels eines Systemkommandos auf der Konsole aufzurufen. In SAS können Systemkommandos über die X-Anweisung oder im SAS Data Step über die System-Routine bzw. -Funktion ausgeführt werden. Allen genannten Varianten ist gemein, dass sie das Benutzerrecht voraussetzen, Systemkommandos ausführen zu dürfen. Der Unterschied liegt darin, dass es sich bei „X“ um eine sogenannte globale SAS-Anweisung handelt, die unabhängig von Bedingungen (im Data Step) ausgeführt wird. Und die System-Routine bzw. -Funktion hingegen im Data Step bedingungsabhängig ausgeführt werden können. Die System-Funktion liefert im Gegensatz zur Routine einen Rückgabewert. Auch mittels dieser Methode sind prinzipiell dieselben Funktionen zum Datentransfer und zur Nutzung von R-Funktionalität ausführbar wie mit der IML-Methode.

Im Unterschied zur IML-Methode erfolgt der Transfer von Daten und Ergebnissen stets über externe Dateien. SAS Datasets werden beispielsweise in Text- oder CVS-Dateien exportiert und anschließend von R durch geeignete Anweisungen importiert. Dieser Transfer kann auch in die andere Richtung erfolgen. Text-Ausgabeergebnisse oder Logbuchausgaben werden ebenfalls aus externen Dateien weiterverarbeitet und innerhalb von SAS entweder im Output- oder im Log-Fenster angezeigt. Grafische Ausgaben können über externe Dateien importiert werden. Der Aufruf der R-Funktionen erfolgt über ein generiertes oder bereits existierendes Skript, welches dem R-System beim Start-Kommando als Parameter übergeben wird.

Beispiel

Im folgenden Programmbeispiel wird wieder die lineare Regression zwischen den Variablen *Weight* und *Height* des SAS Datasets *Sashelp.Class* aufgegriffen und die prinzi-

pielle Anwendungsweise mittels Verwendung eines Systemskommandos gezeigt. Die Abarbeitung erfolgt hierbei in den folgenden 4 Schritten:

1. Schritt: Exportieren des SAS Datasets *Sashelp.Class* in eine externe CSV-Datei *sashelp.class.csv*.

```
proc export data=Sashelp.Class
  outfile="C:\temp\sashelp.class.csv"
  dbms=csv
  replace;
  delimiter=';';
run;
```

2. Schritt: Generieren eines R-Skripts *lmModel.R* für den Datenimport der CSV-Datei *sashelp.class.csv*, das Ausführen der lm-Funktion zur Durchführung der linearen Regression, das Erzeugen des Dataframes *Model* und den Datenexport in die externe CSV-Datei *model.csv*.

```
data _null_;
  file 'C:\temp\lmModel.R';
  put 'Class <- read.table("C:/temp/sashelp.class.csv", sep=";",
       header=TRUE)';
  put 'lmModel <- lm(Weight ~ Height, data=Class)';
  put 'Model <- data.frame(lmModel$model,
       Residuals=lmModel$residuals, Effects=lmModel$effects,
       FittedValues=lmModel$fitted.values)';
  put 'write.table(Model, "C:/temp/model.csv", sep=";",
       row.names=FALSE)';
run;
```

3. Schritt: Ausführen des generierten R-Skripts *lmModel.R*. Ist der Pfad für R.exe in der Umgebungsvariable PATH definiert, so muss er nicht im Aufruf-Kommando spezifiziert sein. Wie eingangs erwähnt, gibt es verschiedene Varianten, R-Skripte auszuführen. Im folgenden Beispiel wird die CALL SYSTEM-Routine verwendet. Bei der Verarbeitung werden Logbuchausgaben in die Datei *lmModel.log* geschrieben.

```
options xwait xsync;
data _null_;
  call system("R.exe --no-save --no-restore < C:\temp\lmModel.R
    > C:\temp\lmModel.log");
run;
```

Sollen auch die Logbuchausgaben der R-Ausführung im SAS LOG-Fenster angezeigt werden, so müssen diese zunächst eingelesen und anschließend im LOG-Fenster ausgegeben werden. Der folgenden SAS Data Step realisiert diese Ausgabe.

```
data _null_;
  infile 'C:\temp\lmModel.log';
  file log;
  input;
  put 'R-Console: ' _infile_;
run;
```

4. Schritt: Am Schluss werden die von R erzeugten Daten in einen SAS Dataset *Model* importiert. Nun können sie mit SAS weiterverarbeitet werden, hier im Beispiel durch die PRINT-Prozedur (vgl. auch Abbildung 1).

```
proc import datafile="C:\temp\model.csv"
  out=Model
  dbms=csv
  replace;
  delimiter=';';
  getnames=yes;
run;
proc print data=Model;
run;
```

Die vier Schritte können zusammen in einem SAS-Programm ausgeführt werden. Sofern keine unvorhersehbaren Ereignisse geschehen, merkt der Anwender keinen nennenswerten Unterschied zur IML-Methode. Gibt es beispielsweise Probleme beim Datentransfer, so müssen diese zunächst gelöst werden, bevor die Verarbeitung fortgesetzt werden kann. Im Allgemeinen gibt es bei größeren Datenbeständen mit mehr Variabilität in den Daten oft Probleme beim Datentransfer, weshalb diese Methode für allgemeine Zwecke nur begrenzt einsetzbar ist.

3.3 Java Object

Das Java Object ist Teil des Data Step Component Interface und seit SAS Version 9.2 verfügbar. Es bildet die Schnittstelle zu außerhalb von SAS liegenden Java-Klassen. Das Java Object ist prinzipiell auch ohne R nutzbar und stellt eine Möglichkeit zum Instanziieren von Java-Klassen sowie zum Zugreifen auf Felder und Methoden der resultierenden Objekte innerhalb von SAS zur Verfügung. Voraussetzung zur Nutzung dieser Methode ist, dass eine JVM (Java Virtual Machine) auf dem betreffenden Rechner installiert ist. Die Instanziierung erfolgt durch das javaobj-Konstrukt[7]. Damit der Zugriff auf eine Java-Klasse durchführbar ist, muss ihre Quelle in der Umgebungsvariablen CLASSPATH definiert sein.

[7] Der implementierte Mechanismus ist ähnlich dem Java Native Interface (JNI).

Zur Nutzung der Java Object-Methode als R-Schnittstelle werden zwei Komponenten benötigt. Zum einen die bereits erwähnte Java-Klasse, die das Bindeglied zwischen SAS und R bildet. Sie stellt auf der einen Seite die Methoden für SAS bereit, um Daten zu transportieren und um R-Funktionen ausführen zu können. Auf der anderen Seite dient diese Java-Klasse zur Kommunikation mit R. Die zweite Komponente ist ein SAS Data Step zur Kommunikation mit der obigen Java-Klasse. Für R selbst gibt es verschiedene Erweiterungen zur Kommunikation auf Basis von Java. Die in diesem Beitrag verwendete Erweiterung ist RServe[8]. Auch mittels dieser Methode sind prinzipiell dieselben Funktionen zum Datentransfer und zur Nutzung von R-Funktionalität ausführbar wie mit der IML-Methode bzw. der Systemkommando-Methode.

Beispiel

In diesem Programmbeispiel wird abermals der SAS Dataset *Sashelp.Class* herangezogen und die prinzipielle Anwendungsweise mittels Verwendung des Java Objects gezeigt. Dazu ist zunächst eine Java-Klasse zu implementieren, mit deren Hilfe die Kommunikation zwischen SAS und R erfolgen soll. Soll diese Java-Klasse ausschließlich für den genannten SAS Dataset *Sashelp.Class* verwendet werden, so gestalten sich bestimmte Aspekte einfacher, als würde die Java-Klasse allgemein für beliebige Datasets verwendet werden müssen[9].

Wenn demzufolge bekannt ist, dass der SAS Dataset *Sashelp.Class* die 5 Variablen *Name*, *Sex*, *Age*, *Height* und *Weight* enthält, so kann eine geeignete Java-Klasse namens *Student* erzeugt werden, um die individuellen Daten der Studenten zu speichern. Die Java-Klasse enthält die Deklaration ihrer Attribute und einen Konstruktor zur Instanziierung.

```
class Student {
    String name;
    String sex;
    double age;
    double height;
    double weight;

    public Student(String _name, String _sex, double _age,
    double _height, double _weight) {
        name   = _name;
        sex    = _sex;
        age    = _age;
        height = _height;
        weight = _weight;
    }
}
```

[8] Weitere Erweiterungen sind beispielsweise JRI (rJava) und RMI.

[9] In der Realität ist die Einsatzfähigkeit einer Java-Klasse ausschließlich für einen bestimmten SAS Dataset sehr begrenzt.

Zur Speicherung aller Studenten wird eine weitere Java-Klasse *Klasse* benötigt. Die Java-Klasse *Klasse* besteht aus einer ArrayList *studenten* für alle 19 Studenten und enthält eine Methode *addStudent()* zur Ergänzung von Studenten.

```
import java.util.ArrayList;
public class Klasse {
    private ArrayList<Student> studenten = new
      ArrayList<Student>(19);

    public void addStudent(String name, String sex, double age,
    double height, double weight) {
        studenten.add(new Student(name, sex, age, height, weight));
    }
}
```

Mit der Methode *addStudent()* können nun alle Datensätze des SAS Datasets *Sashelp.Class* in eine Instanz des Java-Objekts *Klasse* übertragen werden. Damit die in der betreffenden Instanz der Java-Klasse *Klasse* erzeugten Daten mittels R verarbeitet werden können, sind weitere Ergänzungen notwendig, die im Folgenden ohne den Java-Code darzustellen, beschrieben werden[10]. Zum einen wird eine Methode benötigt, um mit R in Verbindung zu treten. In diesem Beispiel erfolgt die Kommunikation mit der Erweiterung RServe[11], die zugehörige Methode sei *initializeEngine()*. Damit die Java-Seite weiß, wann keine Daten mehr übertragen werden und aus der Java-Klasse *Klasse* ein Dataframe erzeugt werden kann, wird eine Methode *createDataFrame()* implementiert. Sie wandelt die ArrayList *studenten* in einen R-Dataframe *Class* um und transportiert diesen vom Java Workspace zum R Workspace. Als weitere Funktion wird die Ausführung von R-Code benötigt. Diese Funktion wird mit der Methode *execute()* realisiert, die als Parameter den R-Code übertragen bekommt. Des Weiteren wird noch eine Methode *getDataFrame()* für den Datentransfer vom R Workspace zum Java Workspace bzw. von Java nach SAS benötigt. Da die Ergebnisdaten nicht mit den Eingangsdaten des Datasets *Sashelp.Class* übereinstimmen, können die Daten der Rückgabe nicht einfach im SAS Data Step verarbeitet werden. Eine Möglichkeit zur Erzeugung des Ergebnis-Datasets *Model* bestände darin, im Data Step SAS-Code zu generieren, der den Dataset erzeugt. Eine andere – etwas elegantere – Variante wäre die Erzeugung des Datasets mittels eines SCL-Programms[12]. Mit der Methode *terminateEngine()* wird die Kommunikation zwischen Java und R getrennt.

[10] Die Darstellung des gesamten Java-Codes würde den Rahmen dieses Beitrags sprengen.

[11] Die Erweiterung RServe stellt eine Client/Server-Kommunikation basierend auf dem TCP/IP-Protokoll bereit. Der Server „RServe“ muss also gestartet sein und bietet dann seine Dienste den Clients an.

[12] SCL-Programme können mittels Base SAS ausgeführt werden, jedoch nur mit Hilfe von SAS/AF implementiert werden. Ein allgemeines SCL-Programm zur Erzeugung beliebiger SAS Datasets nach bestimmten Konventionen kann somit innerhalb von Base SAS ausgeführt werden.

Um mittels des Java Objects aus SAS heraus auf die Methoden der Java-Klassen zugreifen zu können, muss zunächst das Java Object mit Hilfe des Javaobj-Konstrukts instanziiert werden. Dies erfolgt nur einmal beim ersten Durchlauf des SAS Data Steps (_n_=1). Bei jedem Durchlauf des SAS Data Steps wird ein Datensatz (entsprechend einem Student) mit der Methode *addStudent()* in der erzeugten Instanz der Java-Klasse *Klasse* gespeichert. Nach der Übergabe aller Datensätze aus dem SAS Dataset *Sashelp.Class* (if last then do; … end;) wird eine Verbindung zu R aufgebaut (*initializeEngine()*), ein R-Dataframe (*createDataFrame()*) erzeugt, ein R-Skript ausgeführt (*execute()*) und anschließend die Ergebnisdaten nach SAS übertragen und der Ergebnis-Dataset aufgebaut (Makro *%dsbuild*)[13] und die Verbindung zu R getrennt (*terminateEngine()*).

```
data _null_;
  set sashelp.class end=last;
  /* Java Object instanziieren */
  if _n_=1 then declare javaobj j("Klasse");
  /* Studenten Datensatz für Datensatz ergänzen */
  j.callVoidMethod("addStudent", name, sex, age, height, weight);
  /* Wenn alle Datensätze (Studenten) übertragen wurden */
  if last then do;
    /* Verbindung zu R aufbauen */
    j.callVoidMethod("initializeEngine");
    /* R Dataframe erzeugen */
    j.callVoidMethod("createDataFrame", "Class");
    /* R-Skript ausführen */
    j.callVoidMethod("execute",
      "lmModel <- lm(Weight ~ Height, data=Class);
       Model <- data.frame(lmModel$model,
       Residuals=lmModel$residuals,
       Effects=lmModel$effects,
       FittedValues=lmModel$fitted.values)");
    /* R-Dataframe nach SAS holen */
    j.callVoidMethod("getDataFrame", "Model");
    call execute('%dsbuild');
    /* Verbindung zu R trennen */
    j.callVoidMethod("terminateEngine");
  end;
run;
```

Logbuchausgaben können direkt ins SAS LOG-Fenster ausgeben werden, sofern sie mittels Java nach Standard-Output (stdout) geschrieben werden. Die obige Darstellung ist nicht allgemein anwendbar, sondern auf den SAS Dataset *Sashelp.Class* zugeschnit-

[13] Es soll an dieser Stelle nicht darauf eingegangen werden, wie das Makro *%dsbuild* arbeitet, sondern dazu auf Kapitel 4 verwiesen werden.

ten. Für eine allgemeine Schnittestelle zur Verarbeitung beliebiger SAS Datasets ist der Sachverhalt komplizierter, da zunächst stets die Metadaten zur Beschreibung des Aufbaus des zu verarbeitenden SAS Datasets und anschließend die Datenwerte geeignet an die Java-Klasse übergeben werden müssen. Auch auf der Java-Seite müssen die korrespondierenden Methoden universeller implementiert sein.

3.4 Zusammenfassung

Jede der dargestellten Methoden hat Vor- und Nachteile. Die SAS/IML-Methode passt sich am Besten in die SAS-Landschaft ein und ist am einfachsten für den Anwender nutzbar. Allerdings ist das SAS-Produkt IML erforderlich und es gibt auch einige Nachteile, wie die folgende Übersicht in Tabelle 1 zeigt.

Tabelle 1: Vor- und Nachteile der Schnittstellen zu R

Schnittstelle	**Vorteile**	**Nachteile**
SAS/IML	Einfache Handhabung	Produkt SAS/IML erforderlich
	Keine Einschränkung bzgl. R-Funktionen	R-Versionsauswahl teilweise eingeschränkt
	Automatische Datentypumwandlung	Feste Datentypumwandlung, einige Typen nicht nutzbar
	Erweiterte Fehlerbehandlung	Nicht für alle Betriebssysteme verfügbar
Systemkommando	Aus SAS-Sicht einfache Handhabung, da viele Funktionen von R ausgeführt werden	Ausführungsrecht erforderlich
	Kein Zusatzprodukt erforderlich	Medienbrüche beim Datentransfer
		Datentransfer via externer Dateien
Java Object	Kein Zusatzprodukt erforderlich	Abhängig von Fähigkeiten des Java Objects
	Keine externen Dateien für Datentransfer notwendig	Kompliziert für den generischen allgemeinen Fall
	Fehlerbehandlung javaseitig implementierbar	
	Weitere Funktionalität und Datentypumwandlungen implementierbar	

Die Java Object-Methode ist für die allgemeine Anwendbarkeit kompliziert, hat aber viel Potential für Erweiterungen, insbesondere in Hinblick auf Funktionalität und Datentypumwandlungen.

Im nachfolgenden Kapitel wird die Java Object-Methode zu einem allgemeinen generischen Verfahren weiterentwickelt. Damit für den Anwender die Handhabung einfach ist, werden außerdem Möglichkeiten gezeigt, die technische Realisierung zu kapseln. Die verschiedenen notwendigen Komponenten für die Integration werden beschrieben.

4 Allgemeine Integration von R mit Java Object

Sollen nun die Vorteile der Java Object-Methode genutzt und die Nachteile minimiert werden, ist man geneigt, den komplizierten Mechanismus für den Anwender zu verbergen. Hierzu bietet sich die FCMP-Prozedur hervorragend an. Einen allgemeinen generischen Ansatz erreicht man durch Implementierung allgemeiner generischer Java-Klassen und einer Erweiterung des SAS-seitigen Datentransfers.

Allgemein erfolgt die Abarbeitung hierbei mittels entsprechender Methoden in den in Tabelle 2 gezeigten drei Schritten:

Tabelle 2: Allgemeine Schritte bei Verwendung der Java Object-Methode

Schritt	**Aktion**
1. Schritt:	Datentransfer von SAS zu Java
2. Schritt:	Verbindung zu R aufbauen
	Dataframe erzeugen und Datentransfer von Java zu R
	R-Funktion ausführen (R-Code, R-Skript)
	Datentransfer von R zu Java
	Verbindung zu R trennen
3. Schritt:	Datentransfer von Java zu SAS

Zur Realisierung bedarf es für die bereits in Abschnitt 3.3 beschriebenen Methoden verallgemeinerte Entsprechungen.

4.1 Implementierung

Neben dem bereits vorgestellten Java Object im SAS Data Step werden für die zu implementierende Schnittstelle weitere Werkzeuge benötigt. Innerhalb von SAS werden die FCMP-Prozedur, ein SAS Makro und ein SCL-Programm verwendet. Außerhalb von SAS werden allgemeine Java-Klassen benötigt, die beliebige Daten zwischen SAS und R transferieren sowie R-Code an R zur Ausführung übergeben können. In der Abbildung 2 wird die Systemarchitektur unter Verwendung der genannten Komponenten gezeigt. Die einzelnen Komponenten werden in den folgenden Abschnitten beschrieben.

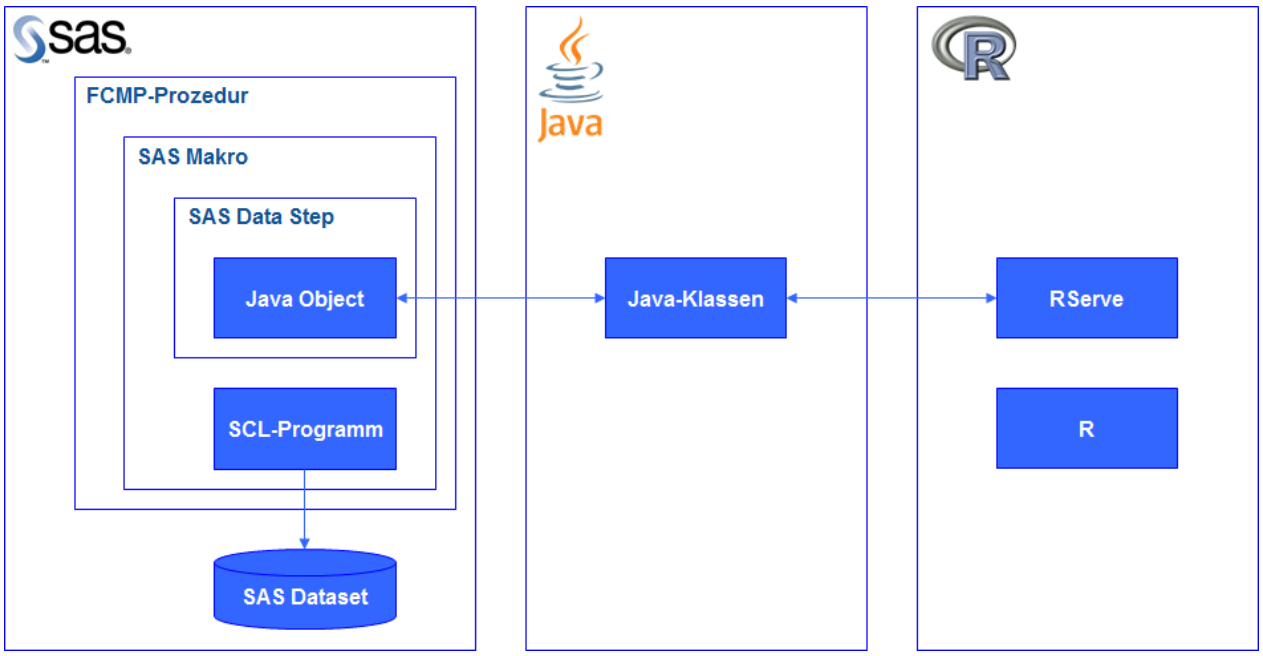

Abbildung 2: Systemarchitektur

4.1.1 SAS Data Step, Java Object und Java-Klassen

Der SAS Data Step bildet den inneren Rahmen der Schnittstelle. Innerhalb des Data Steps werden das Java Object instanziiert[14], die an R zu transferierenden Daten aus dem gewählten SAS Dataset ausgelesen und elementweise an Java übergeben. Dabei ist die allgemeine Methode der speziellen in Abschnitt 3.3 sehr ähnlich, nur dass statt der Methode *addStudent()* mit den 5 Parametern für die Variablen des SAS Dataset *Sashelp.Class* eine allgemeine Methode *addElement()* verwendet wird. Die Methode *addElement()* hat die 3 Parameter *type*, *value* und *rownumber*. Ist *rownumber* = 0, so werden die Metadaten des betreffenden SAS Datasets übergeben, andernfalls entspricht *rownumber* dem Datensatz (bzw. der Beobachtung), ist *rownumber* = -1, wurde der SAS Dataset vollständig übertragen. Eine Erweiterung beispielsweise hinsichtlich der Übergabe von Format-Informationen, wäre leicht dadurch zu realisieren, dass die entsprechenden Informationen ebenfalls über diese Methode an die Java-Klasse weitergereicht und dort geeignet verarbeitet werden. Die Methoden *initializeEngine()* und *terminateEngine()* sowie *createDataFrame()* und *execute()* können, wie im Abschnitt 3.3 beschrieben, verwendet werden. Weiterhin werden noch Methoden für den Datentransfer von R nach SAS benötigt. Die Methode *getDataFrame()* wählt den zu übertragenden R-Dataframe im R Workspace aus und generiert ein entsprechendes Java-Objekt. Mittels der Methode *getDfDesc()* werden die Metadaten des generierten Java-Objekts übermittelt. Und die Methode *getNext()* liefert solange Datenwerte, bis die gesamten Datenzeilen ausgelesen wurden, dann ist der Rückgabewert leer.

[14] Vgl. auch http://support.sas.com/rnd/base/datastep/dot/javaobj.html.

Das folgende Code-Fragment zeigt die elementweise Übergabe der Daten des SAS Datasets nach R. Zunächst werden die Metadaten des SAS Datasets übergeben und anschließend jeder Datensatz Feld für Feld. Da jeder Datenwert als Zeichenfolge übergeben wird, muss zusätzlich der betreffende Datentyp übergeben werden. Entsprechend der SAS-Konventionen sind dies entweder numerische („N“) oder alphanumerische („C“) Datenwerte.

```
/*** SAS Input Dataset an R übergeben ***/
if symget("input") ne "" then do;
  dsid = open("&input");
  if _dsid then do;
    num = attrn(_dsid,'nvars');
    do _i = 1 to _num;
      j.callVoidMethod("addElement", vartype(_dsid,_i),
        varname(_dsid,_i), 0);
    end;
    rc=FETCH(_dsid);
    _rownumber+1;
    do while (rc ne -1);
      do _i=1 to _num;
        if vartype(_dsid,_i) = "C" then
          j.callVoidMethod("addElement", "C",
            GETVARC(_dsid,varnum(_dsid,varname(_dsid,_i))),
            _rownumber);
        else if vartype(_dsid,_i) = "N" then
          j.callVoidMethod("addElement", "N",
            put(GETVARN(_dsid,varnum(_dsid,varname(_dsid,_i))),
            best.),_rownumber);
      end;
      rc=FETCH(_dsid);
      _rownumber+1;
    end;
    j.callVoidMethod("addElement", "X", "X", -1);
    rc=close(_dsid);
  end;

  /*** R Dataframe erzeugen ***/
  j.callVoidMethod("createDataFrame", "&input");
end;
```

4.1.2 SCL-Programm (Ausführung)

Aufgabe des SCL-Programms *dsbuild.scl* ist die Erzeugung eines neuen weiteren SAS Datasets innerhalb eines Data Steps, dessen Struktur erst zur Ausführungszeit ermittelt werden kann. Das Programm hat zwei Parameter. Der erste Parameter legt den Modus

fest, in der das SCL-Programm arbeitet. Der zweite Parameter enthält die Werte für den neuen SAS Dataset. Es gibt zwei Modi. Der eine Modus ist „M“ und legt fest, dass die Werte Metadaten zum Aufbau der Struktur des SAS Datasets sind. Der andere Modus ist „D“ und legt fest, dass die Werte Datenwerte des SAS Datasets sind. Das folgende Code-Fragment zeigt die Aufrufe des SCL-Programms *dsbuild.scl* (mittels des Makros `%dsbuild`) in den beiden Modi.

```
/* R-Dataframe zeilenweise nach SAS holen */
if symget("output") ne "" then do;
  j.callVoidMethod("getDataFrame", "&output");
  j.callStringMethod("getDfDesc", dfDesc);
  call execute('%dsbuild(%quote([M]'||trim(left(dfDesc))||'))');
  j.callStringMethod("getNext",datavalues);
  do while(datavalues ne "");
    call execute('%dsbuild(%nrbquote([D]'||trim(left(datavalues))
      ||'))');
    j.callStringMethod("getNext",datavalues);
  end;
end;
```

4.1.3 SAS Makro und FCMP-Prozedur

Die Aufgabe des SAS-Makros ist lediglich technischer Natur und notwendig, damit der gesamte innere SAS Data Step (vgl. Abschnitt 4.1.1) in die FCMP-Definition integriert werden kann, was anderenfalls nicht erlaubt wäre.

Die FCMP-Prozedur ist eine SAS Base-Prozedur zum Definieren von Funktionen und Call-Routinen, die sich im DATA Step oder im PROC Step verwenden lassen. Mit Hilfe der FCMP-Prozedur kann also die Palette der SAS-Funktionen und -Routinen erweitert werden. In diesem Beitrag wird mittels der FCMP-Prozedur die durch die Schnittstelle zu R angebotene Funktionalität gekapselt. Es entsteht eine Call-Routine namens *Rexec()*. Die gesamte Schnittstelle wird innerhalb der Routinen-Definition bzw. des Makros „verborgen“.

```
proc fcmp outlib=sasuser.functions.rex;
   subroutine rexec(input$, output$, rscript$);
      javaclass = 'sas_r_rserve';
      rc = run_macro('rexec', javaclass, input, output, rscript);
   endsub;
quit;
```

4.2 Anwendung

Als Benutzerschnittstelle zur Ausführung des Datentransfers und der gewünschten R-Funktionalität wird die CALL-Routine *Rexec()* aufgerufen:

```
data _null_;
  weitere Anweisungen
  call Rexec(<INPUT>, <OUTPUT>, <R-Skript>);
  weitere Anweisungen
run;
```

Die CALL-Routine *Rexec()* hat drei Parameter. Der 1. Parameter ist der Name des INPUT-Datasets und des korrespondierenden R-Dataframes, der 2. Parameter ist der Name des OUTPUT-Datasets (und des entsprechenden R-Dataframes) und der 3. Parameter ist der Name einer R-Skriptdatei oder ein R-Skript (R-Code, wie im folgenden Beispiel).

Beispiel

Das folgende Programmbeispiel zeigt die Anwendungsweise mittels der CALL-Routine *Rexec()* abermals unter Verwendung des SAS Datasets *Sashelp.Class* und der Ausführung einer einfachen linearen Regression zwischen den Variablen *Weight* und *Height*.

```
data _null_;
   call Rexec('Sashelp.Class', 'Model',
              'lmModel <- lm(Weight ~ Height, data=Sashelp.Class);
               Model <- data.frame(lmModel$model,
               Residuals=lmModel$residuals,Effects=lmModel$effects,
               FittedValues=lmModel$fitted.values)');
run;
proc print data=Model;
run;
```

Die Ausgabe der PRINT-Prozedur entspricht wieder der aus dem Abschnitt 3.1 (vgl. Abbildung 1).

5 Zusammenfassung

In den vorangegangenen Kapiteln wurde gezeigt, dass es verschiedenen Methoden gibt, um R-Funktionalität innerhalb von SAS nutzen zu können. Es wurde weiterhin beschrieben, wie mit Hilfe des Java Objects eine eigene Schnittstelle zu R implementiert werden kann und wie die möglicherweise komplizierte Implementierung durch Verwendung der FCMP-Prozedur für den Benutzer leicht gehandhabt werden kann.

Literatur

[1] A. Hilbert, R. Minkenberg (Hrsg.): Proceedings der 16. Konferenz der SAS®-Anwender in Forschung und Entwicklung (KSFE). Shaker-Verlag, 2012.

Weiterführende Literatur

- Base SAS® 9.2 Procedures Guide. Cary, NC: SAS Institute Inc. 2009.
- SAS® 9.2 Language Reference: Dictionary, Fourth Edition. Cary, NC: SAS Institute Inc. 2011.
- SAS/IML® 9.22 User's Guide. Cary, NC: SAS Institute Inc. 2010.
- Ken Kleinman: SAS and R: data management, statistical analysis, and graphics / K. Kleinman and N. J. Horton. Taylor and Francis Group, LLC. 2010.