Mayday RLink – The best of both worlds

Florian Battke, Stephan Symons, Kay Nieselt

battke@informatik.uni-tuebingen.de

July 8, 2009

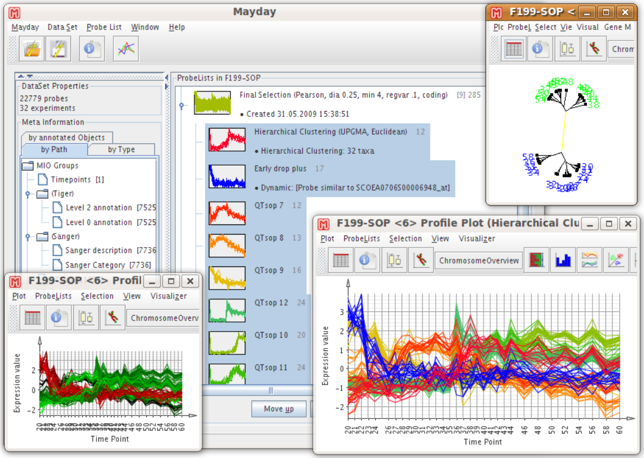
Eberhard Karls Universität Tübingen

Zentrum für BioInformatik Tübingen

PAS
Proteomics Algorithmen und Simulation

Outline

Mayday – An extensible visualization platform



- Basic data structure is a numeric matrix
  - columns are observations, rows are “features” of interest
  - Aim is to find (full-width) submatrices with common features

Mayday – An extensible visualization platform

Strengths

- Cross-platform: Written in **Java**
- Structured display of submatrices
- **Plugin-based** $\rightarrow$ fast integration of new methods
- **Interactive** visualizations, different views are linked
- Visualizations can be *enhanced* by **meta-data**
- Focus: **visual** data exploration and hypothesis generation

One big deficit

No live programmers' access to the data.
$\rightarrow$ "Power-users" often need to move data to R and back

Mayday – An extensible visualization platform

Strengths

- Cross-platform: Written in **Java**
- Structured display of submatrices
- **Plugin-based** → fast integration of new methods
- **Interactive** visualizations, different views are linked
- Visualizations can be *enhanced* by **meta-data**
- Focus: **visual** data exploration and hypothesis generation

One big deficit

No live programmers’ access to the data.
→ “Power-users” often need to move data to R and back

Requirements

Integration of an interactive R shell into Mayday

- Live access to Mayday's data
- Efficient data management
- Memory-safe data manipulation
- Objects behave as much like real R objects as possible

Possible solutions

Self-made interface

e.g. using pipes

+ no process limit
+ could be interactive

– slow
– a LOT of work

RServe / RSJava

using sockets

+ no process limit
+ no direct dependency

– Java accessing R
– no interactive session
– still lots of work

JRI + RJava

R embedded in JVM

– only one R instance

+ shared memory
+ R accessing Java
+ interactivity built in
+ very fast

Short overview: JRI+RJava

- Using Java objects in R: rJava
- Embedding R in Java: JRI
- One process (JVM), memory shared between VM and R
- R event loop waiting for input from Java callbacks

Possible solutions

Self-made interface

e.g. using pipes
+ no process limit
+ could be interactive

– slow
– a LOT of work

RServe / RSJava

using sockets
+ no process limit
+ no direct dependency
– Java accessing R
– no interactive session
– still lots of work

JRI + RJava

R embedded in JVM
– only one R instance
+ shared memory
+ R accessing Java
+ interactivity built in
+ very fast

Short overview: JRI+RJava

- Using Java objects in R: rJava
- Embedding R in Java: JRI
- One process (JVM), memory shared between VM and R
- R event loop waiting for input from Java callbacks

Possible solutions

Self-made interface

e.g. using pipes

+ no process limit
+ could be interactive

– slow
– a LOT of work

RServe / RSJava

using sockets

+ no process limit
+ no direct dependency
– Java accessing R
– no interactive session
– still lots of work

JRI + RJava

R embedded in JVM

– only one R instance
+ shared memory
+ R accessing Java
+ interactivity built in
+ very fast

Short overview: JRI+RJava

- Using Java objects in R: rJava
- Embedding R in Java: JRI
- One process (JVM), memory shared between VM and R
- R event loop waiting for input from Java callbacks

Possible solutions

Self-made interface

e.g. using pipes
- \+ no process limit
- \+ could be interactive
- – slow
- – a LOT of work

RServe / RSJava

using sockets
- \+ no process limit
- \+ no direct dependency
- – Java accessing R
- – no interactive session
- – still lots of work

JRI + RJava

R embedded in JVM
- – only one R instance
- \+ shared memory
- \+ R accessing Java
- \+ interactivity built in
- \+ very fast

Short overview: JRI+RJava

- Using Java objects in R: rJava
- Embedding R in Java: JRI
- One process (JVM), memory shared between VM and R
- R event loop waiting for input from Java callbacks

Some thoughts on memory management

Pointers

- no copying needed
- very fast
- uncontrolled access
- GC issues

Copied objects

- slow
- memory-intensive
- controlled access
- hard too keep in sync

"Controlled references"

- Leightweight S3 objects, containing
  - Identifier (integer), used by Java as object reference
  - Type/Class (string), used by R to resolve function calls
- copy data as needed, still very fast
- Java program decides what to expose to R

Some thoughts on memory management

Pointers

- no copying needed
- very fast
- uncontrolled access
- GC issues

Copied objects

- slow
- memory-intensive
- controlled access
- hard too keep in sync

"Controlled references"

- Leightweight S3 objects, containing
  - Identifier (integer), used by Java as object reference
  - Type/Class (string), used by R to resolve function calls
- copy data as needed, still very fast
- Java program decides what to expose to R

Some thoughts on memory management

Pointers

- no copying needed
- very fast
- uncontrolled access
- GC issues

Copied objects

- slow
- memory-intensive
- controlled access
- hard too keep in sync

“Controlled references”

- Leightweight S3 objects, containing
  - Identifier (integer), used by Java as object reference
  - Type/Class (string), used by R to resolve function calls
- copy data as needed, still very fast
- Java program decides what to expose to R

Thoughts on user-friendliness

Fetching a value from a `HashMap<String, Integer>`

- **JAVA**

```
int ret = hashMap.get("Key")
```

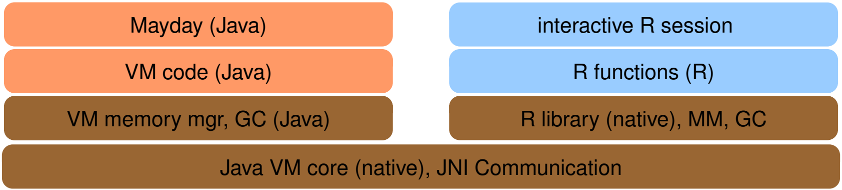
- **native rJava**

```
key <- .jnew( "Ljava/lang/String;", "Key" );
ret <- .jcall( hashMap,
               "Ljava/lang/Object;",
               "get",
               .jcast(key, "Ljava/lang/Object")
ret <- .jcast( ret, "Ljava/lang/Integer" );
ret <- .jcall( ret, "I", "intValue" );
```

- **Our aim for RLink**

```
ret <- hashMap[["Key"]]
```

Command translation and data flow



One object “**ref**” is shared between Mayday and R

Example: `(int) ret ← hashMap[["Key"]]` with class “rlink.hm” and id “5”

1. R resolves operator [[for class “rlink.hm”

Command translation and data flow

Mayday (Java)	interactive R session
VM code (Java)	R functions (R)
VM memory mgr, GC (Java)	R library (native), MM, GC
Java VM core (native), JNI Communication	

One object “**ref**” is shared between Mayday and R

Example: `(int) ret ← hashMap[["Key"]]` with class “rlink.hm” and id “5”

1. R resolves operator **[[** for class “rlink.hm”
2. `[[.rlink.hm(hashMap, "Key")` uses rJava:
   `.jcall(ref, "hmget", 5, .jnew("Ljava/lang/String","Key"))`
3. rJava/JRI transfer
4. `ref.hmget(5,"Key")` resolves “5” to an actual object o,
   calls `o.get("Key")` and packages the return value
5. rJava/JRI transfer
6. `[[.rlink.hm(hashMap, "Key")` unpacks the return value
   and uses rJava functions to convert to a native type (or another “wrapped” object)

Command translation and data flow

Mayday (Java) | interactive R session
VM code (Java) | R functions (R)
VM memory mgr, GC (Java) | R library (native), MM, GC
Java VM core (native), JNI Communication

One object “**ref**” is shared between Mayday and R

Example: `(int) ret ← hashMap[["Key"]]` with class “rlink.hm” and id “5”

1. R resolves operator **[[** for class “rlink.hm”
2. `[[.rlink.hm(hashMap, "Key")` uses rJava:
   `.jcall(ref, "hmget", 5, .jnew("Ljava/lang/String","Key"))`
3. rJava/JRI transfer
4. `ref.hmget(5,"Key")` resolves “5” to an actual object o,
   calls `o.get("Key")` and packages the return value
5. rJava/JRI transfer
6. `[[.rlink.hm(hashMap, "Key")` unpacks the return value
   and uses rJava functions to convert to a native type (or another “wrapped” object)

Command translation and data flow

Mayday (Java)	interactive R session
VM code (Java)	**R functions (R)**
VM memory mgr, GC (Java)	R library (native), MM, GC

Java VM core (native), JNI Communication

One object “**ref**” is shared between Mayday and R

Example: `(int) ret ← hashMap[["Key"]]` with class “rlink.hm” and id “5”

1. R resolves operator **[[** for class “rlink.hm”
2. `[[.rlink.hm(hashMap, "Key")` uses rJava:
   `.jcall(ref, "hmget", 5, .jnew("Ljava/lang/String","Key"))`
3. rJava/JRI transfer
4. `ref.hmget(5,"Key")` resolves “5” to an actual object o,
   calls `o.get("Key")` and packages the return value
5. rJava/JRI transfer
6. `[[.rlink.hm(hashMap, "Key")` unpacks the return value
   and uses rJava functions to convert to a native type (or another “wrapped” object)

Command translation and data flow

Mayday (Java)	interactive R session
VM code (Java)	R functions (R)
VM memory mgr, GC (Java)	R library (native), MM, GC

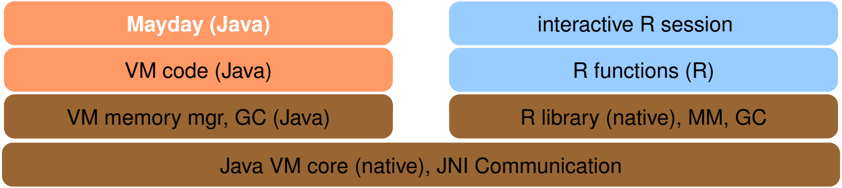
Java VM core (native), JNI Communication

One object “**ref**” is shared between Mayday and R

Example: `(int) ret ← hashMap[["Key"]]` with class “rlink.hm” and id “5”

1. R resolves operator **[[** for class “rlink.hm”
2. `[[.rlink.hm(hashMap, "Key")` uses rJava:
   `.jcall(ref, "hmget", 5, .jnew("Ljava/lang/String","Key"))`
3. rJava/JRI transfer
4. `ref.hmget(5,"Key")` resolves “5” to an actual object o,
   calls `o.get("Key")` and packages the return value
5. rJava/JRI transfer
6. `[[.rlink.hm(hashMap, "Key")` unpacks the return value
   and uses rJava functions to convert to a native type (or another “wrapped” object)

Command translation and data flow



One object "**ref**" is shared between Mayday and R

Example: `(int) ret ← hashMap[["Key"]]` with class "rlink.hm" and id "5"

1. R resolves operator **[[** for class "rlink.hm"
2. `[[.rlink.hm(hashMap, "Key")` uses rJava:
   `.jcall(ref, "hmget", 5, .jnew("Ljava/lang/String","Key"))`
3. rJava/JRI transfer
4. `ref.hmget(5,"Key")` resolves "5" to an actual object o,
   calls `o.get("Key")` and packages the return value
5. rJava/JRI transfer
6. `[[.rlink.hm(hashMap, "Key")` unpacks the return value
   and uses rJava functions to convert to a native type (or another "wrapped" object)

Command translation and data flow

Mayday (Java)	interactive R session
VM code (Java)	R functions (R)
VM memory mgr, GC (Java)	**R library (native), MM, GC**
Java VM core (native), JNI Communication	

One object "**ref**" is shared between Mayday and R

Example: `(int) ret ← hashMap[["Key"]]` with class "rlink.hm" and id "5"

1. R resolves operator **[[** for class "rlink.hm"
2. `[[.rlink.hm(hashMap, "Key")` uses rJava:
   `.jcall(ref, "hmget", 5, .jnew("Ljava/lang/String","Key"))`
3. rJava/JRI transfer
4. `ref.hmget(5,"Key")` resolves "5" to an actual object o,
   calls `o.get("Key")` and packages the return value
5. rJava/JRI transfer
6. `[[.rlink.hm(hashMap, "Key")` unpacks the return value
   and uses rJava functions to convert to a native type (or another "wrapped" object)

Command translation and data flow

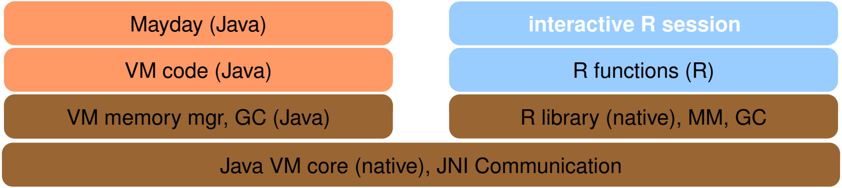
Mayday (Java)	interactive R session
VM code (Java)	**R functions (R)**
VM memory mgr, GC (Java)	R library (native), MM, GC
Java VM core (native), JNI Communication	

One object “**ref**” is shared between Mayday and R

Example: `(int) ret ← hashMap[["Key"]]` with class “rlink.hm” and id “5”

1. R resolves operator **[[** for class “rlink.hm”
2. `[[.rlink.hm(hashMap, "Key")` uses rJava:
   `.jcall(ref, "hmget", 5, .jnew("Ljava/lang/String","Key"))`
3. rJava/JRI transfer
4. `ref.hmget(5,"Key")` resolves “5” to an actual object o,
   calls `o.get("Key")` and packages the return value
5. rJava/JRI transfer
6. `[[.rlink.hm(hashMap, "Key")` unpacks the return value
   and uses rJava functions to convert to a native type (or another “wrapped” object)

Command translation and data flow



One object “**ref**” is shared between Mayday and R

Example: `(int) ret ← hashMap[["Key"]]` with class “rlink.hm” and id “5”

1. R resolves operator **[[** for class “rlink.hm”
2. `[[.rlink.hm(hashMap, "Key")` uses rJava:
   `.jcall(ref, "hmget", 5, .jnew("Ljava/lang/String","Key"))`
3. rJava/JRI transfer
4. `ref.hmget(5,"Key")` resolves “5” to an actual object o,
   calls `o.get("Key")` and packages the return value
5. rJava/JRI transfer
6. `[[.rlink.hm(hashMap, "Key")` unpacks the return value
   and uses rJava functions to convert to a native type (or another “wrapped” object)

Operations of interest

- **All objects**
  - summary, print
- **List-like objects**
  - length
  - names, names←
  - **[[** (select) and **[[**← (replace)
  - **[** (sublist)
  - lapply, sapply
- **Matrix-like objects**
  - nrow, ncol, dim
  - rownames, colnames, rownames←, colnames←
  - **[** (submatrix) and **[**← (replace)
  - apply
- ... and object-specific methods

Overloading depends on context

⇒ We do it dynamically

Operations of interest

- **All objects**
  - summary, print
- **List-like objects**
  - length
  - names, names←
  - **[[** (select) and **[[**← (replace)
  - **[** (sublist)
  - lapply, sapply
- **Matrix-like objects**
  - nrow, ncol, dim
  - rownames, colnames, rownames←, colnames←
  - **[** (submatrix) and **[**← (replace)
  - apply
- ... and object-specific methods

Overloading depends on context

⇒ We do it dynamically

Operations of interest

- **All objects**
  - summary, print
- **List-like objects**
  - length
  - names, names←
  - **[[** (select) and **[[**← (replace)
  - **[** (sublist)
  - lapply, sapply
- **Matrix-like objects**
  - nrow, ncol, dim
  - rownames, colnames, rownames←, colnames←
  - **[** (submatrix) and **[**← (replace)
  - apply
- ... and object-specific methods

Overloading depends on context

⇒ We do it dynamically

Operations of interest

- **All objects**
  - summary, print
- **List-like objects**
  - length
  - names, names←
  - **[[** (select) and **[[**← (replace)
  - **[** (sublist)
  - lapply, sapply
- **Matrix-like objects**
  - nrow, ncol, dim
  - rownames, colnames, rownames←, colnames←
  - **[** (submatrix) and **[**← (replace)
  - apply
- ... and object-specific methods

Overloading depends on context

⇒ We do it dynamically

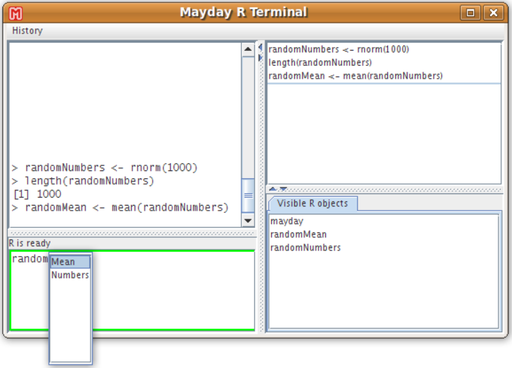
Operations of interest

- **All objects**
  - summary, print
- **List-like objects**
  - length
  - names, names←
  - **[[** (select) and **[[**← (replace)
  - **[** (sublist)
  - lapply, sapply
- **Matrix-like objects**
  - nrow, ncol, dim
  - rownames, colnames, rownames←, colnames←
  - **[** (submatrix) and **[**← (replace)
  - apply
- ... and object-specific methods

Overloading depends on context

⇒ We do it dynamically

Mayday's R terminal

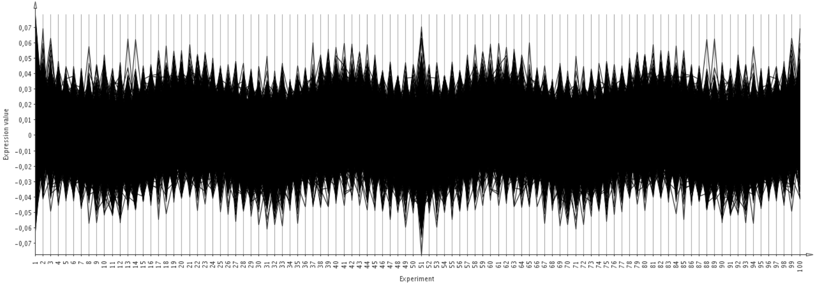


- Multi-line editor
  - syntax highlighting
  - auto-completion
  - brace matching
- History
  - multi-line entries
  - storable
- Live list of user objects

Example

Simulated data:

- 3000 rows (probes), 100 columns
- 1000 probes with random oscillations
- 1000 probes each for two different frequencies



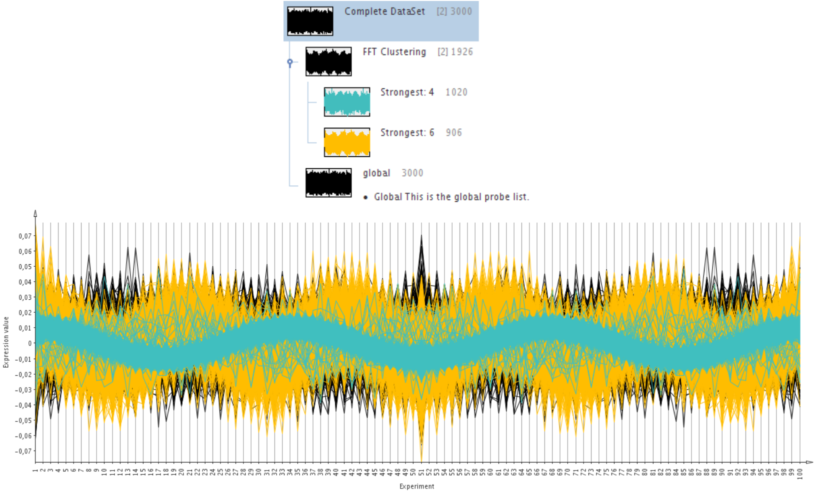
Example (2)

```
TestData <- mayday[["Example"]];                                # <<-- get reference from Mayday
submatrix <- TestData[["Complete DataSet"]]                     # <<-- select submatrix reference
clusterByFFT( submatrix , 50 );

clusterByFFT <- function( probelist , minsize=10 ,
                          parentName="FFT Clustering", prefix="Strongest:" ) {

  f <- probelist[,T]                                            # <<-- extract submatrix

  # perform fft on each row-vector, find strongest factor
  f.fft<-Mod(t(apply(f,1,fft)))
  f.fftrank<-t(apply(-f.fft[,-1],1,rank, ties="first"))
  f.fftrankbest<-apply(f.fftrank,1,
                     function(i) which(i==1)+1)

  ds <- getDataSet( probelist );
  group <- addProbeListGroup(ds, parentName, probelist);        # <<-- create hierarchical stucture

  factors <- unique(f.fftrankbest);
  clusters <- sapply(factors, function(factor) {
    cluster_i <- names(which(f.fftrankbest==factor))
    if (length(cluster_i)>minsize) {
      name <- paste(prefix,factor)
      return (addProbeList(ds, name, cluster_i, group));        # <<-- add a new cluster to Mayday
    }
    return(-1);
  })

  # color the results nicely
  clusters <- clusters[clusters>-1];
  callPlugin( ds, "PAS.core.RecolorProbelists", clusters );     # <<-- call another Mayday plugin
  invisible();
}
```

Example (2)

```
TestData <- mayday[["Example"]];                              # <<-- get reference from Mayday
submatrix <- TestData[["Complete DataSet"]]                   # <<-- select submatrix reference
clusterByFFT( submatrix , 50 );

clusterByFFT <- function( probelist , minsize=10 ,
                          parentName="FFT Clustering", prefix="Strongest:" ) {

  f <- probelist[,T]                                          # <<-- extract submatrix

  # perform fft on each row-vector, find strongest factor
  f.fft<-Mod(t(apply(f,1,fft)))
  f.fftrank<-t(apply(-f.fft[,-1],1,rank, ties="first"))
  f.fftrankbest<-apply(f.fftrank,1,
                      function(i) which(i==1)+1)

  ds <- getDataSet( probelist );
  group <- addProbeListGroup(ds, parentName, probelist);      # <<-- create hierarchical stucture

  factors <- unique(f.fftrankbest);
  clusters <- sapply(factors, function(factor) {
    cluster_i <- names(which(f.fftrankbest==factor))
    if (length(cluster_i)>minsize) {
      name <- paste(prefix,factor)
      return (addProbeList(ds, name, cluster_i, group));      # <<-- add a new cluster to Mayday
    }
    return(-1);
  })

  # color the results nicely
  clusters <- clusters[clusters>-1];
  callPlugin( ds, "PAS.core.RecolorProbelists", clusters );   # <<-- call another Mayday plugin
  invisible();
}
```

Example (3)


Complete DataSet [2] 3000
FFT Clustering [2] 1926
Strongest: 4 1020
Strongest: 6 906
global 3000
Global This is the global probe list.
Expression value
Experiment

Further wishes

- separation of Java and R at the process level
- parallel R instances
- network transparency
- complex R calculations on dedicated machines

Possible solution

Adding an RMI layer → Very few changes needed.

Further wishes

- separation of Java and R at the process level
- parallel R instances
- network transparency
- complex R calculations on dedicated machines

Possible solution

Adding an RMI layer → Very few changes needed.

Summary

- Integration of R and Mayday
- Wrapped Java objects behave like native R objects
- Controlled interface between Mayday and R
- Mode of communication can be changed easily
- Very user-friendly R shell

Mayday is freely available at
http://microarray-analysis.org/

Directions for future work

What we can do

- Generic framework for object wrapping
- Register R functions into Mayday's plugin manager
- Make more Mayday plugins available in R
- use R to script Mayday

Nice to have

- Multithreaded R core
- More crash-resistant JRI

Directions for future work

What we can do

- Generic framework for object wrapping
- Register R functions into Mayday's plugin manager
- Make more Mayday plugins available in R
- use R to script Mayday

Nice to have

- Multithreaded R core
- More crash-resistant JRI

Acknowledgements

The Mayday team

The R developers

The rJava/JRI developers

The Federal Ministry of Education and Research

Federal Ministry
of Education
and Research

Mayday RLink – The best of both worlds

Florian Battke, Stephan Symons, Kay Nieselt

battke@informatik.uni-tuebingen.de

http://microarray-analysis.org/

Eberhard Karls Universität Tübingen

ZBIT Zentrum für BioInformatik Tübingen

PAS Proteomics Algorithmen und Simulation

Operator overloading

Creating overloaded method "X" for objects of class "C" depends on existing definitions of "X".

No previous definition for X
X is primitive
X is an S3 method

$\Downarrow$

new S3 method: X.C()

X is an S4 method

$\Downarrow$

new S4 method for C

We automatically determine which is needed

$\Rightarrow$ functions are built dynamically

Operator overloading

Creating overloaded method “X” for objects of class “C” depends on existing definitions of “X”.

No previous definition for X X is primitive X is an S3 method	X is an S4 method
$\Downarrow$	$\Downarrow$
new S3 method: X.C()	new S4 method for C

We automatically determine which is needed

$\Rightarrow$ functions are built dynamically

Operator overloading

Creating overloaded method “X” for objects of class “C” depends on existing definitions of “X”.

No previous definition for X
X is primitive
X is an S3 method

$\Downarrow$

new S3 method: X.C()

X is an S4 method

$\Downarrow$

new S4 method for C

We automatically determine which is needed

$\Rightarrow$ functions are built dynamically

Operator overloading

Creating overloaded method “X” for objects of class “C” depends on existing definitions of “X”.

No previous definition for X X is primitive X is an S3 method	X is an S4 method
$\Downarrow$	$\Downarrow$
new S3 method: X.C()	new S4 method for C

We automatically determine which is needed

$\Rightarrow$ functions are built dynamically

Limitations

Shared process

- limits memory on 32 bit systems
- Makes JVM vulnerable to crashes in R code
- only one instance of R at a time
- blocking, no parallel execution

Installation

- Requires C and Java compilers, R headers
- Superuser privileges needed
- Can't easily be automated
- So far not working on MacOS with 64 bit Java

Limitations

Shared process

- limits memory on 32 bit systems
- Makes JVM vulnerable to crashes in R code
- only one instance of R at a time
- blocking, no parallel execution

Installation

- Requires C and Java compilers, R headers
- Superuser privileges needed
- Can’t easily be automated
- So far not working on MacOS with 64 bit Java

RMI Connections

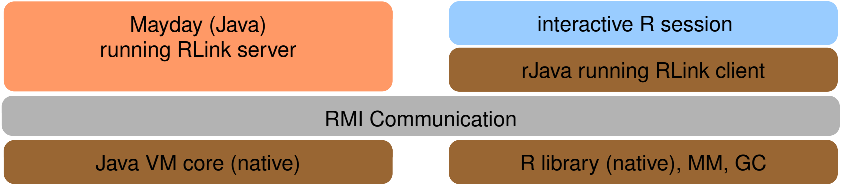
We can easily replace the connection between Mayday and R.

Mayday (Java) running RLink server	interactive R session / rJava running RLink client
RMI Communication	
Java VM core (native)	R library (native), MM, GC

+ Multiple parallel instances
+ Unlimited memory
+ More stable
+ Installation is much simpler
– More work to start a session
– Somewhat slower

RMI Connections

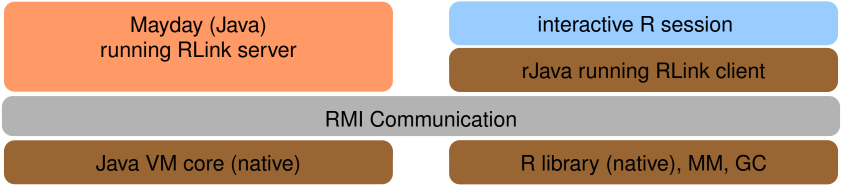
We can easily replace the connection between Mayday and R.



+ Multiple parallel instances
+ Unlimited memory
+ More stable
+ Installation is much simpler
– More work to start a session
– Somewhat slower

RMI Connections

We can easily replace the connection between Mayday and R.



+ Multiple parallel instances
+ Unlimited memory
+ More stable
+ Installation is much simpler
– More work to start a session
– Somewhat slower