MiniProject #1

codename: PimpMyPipe

Rieder Mathias

Jin Yili

Lunev Igor

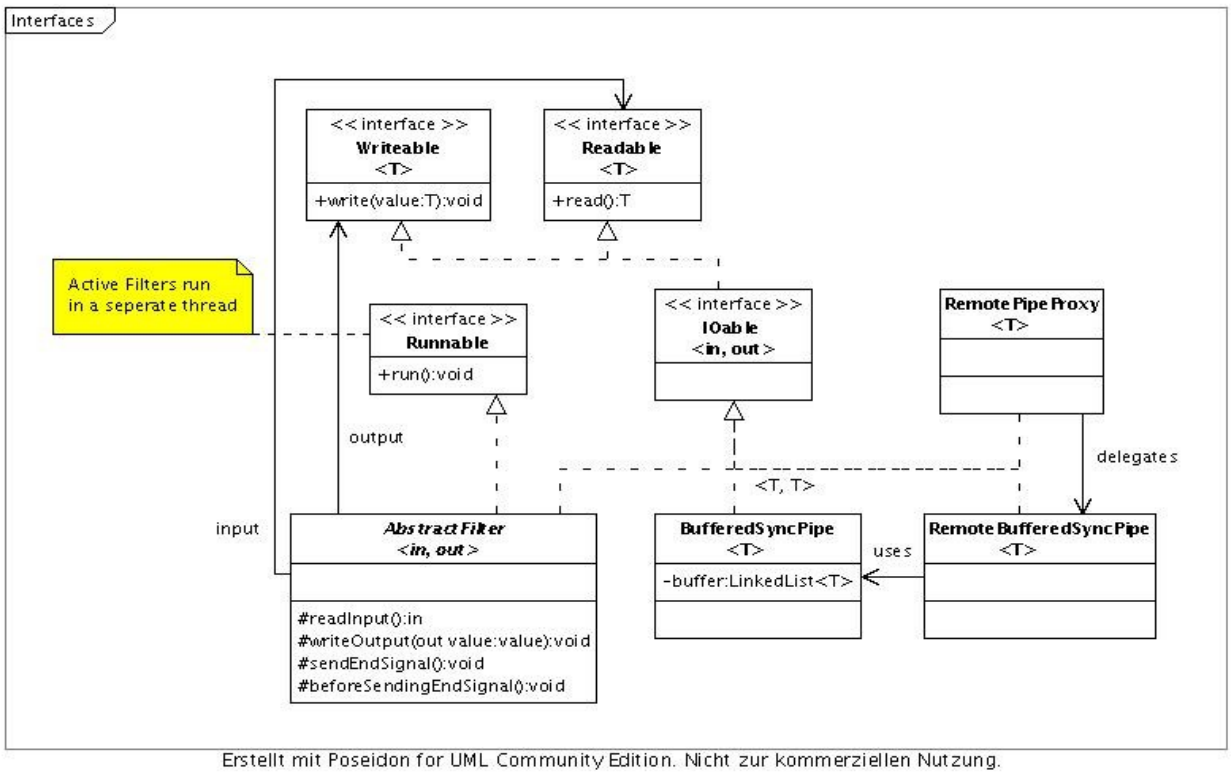
Tuomo Tikkinen

Design goals

- only work with interface
  If you only referencing interfaces instead of the implementing class, you can easily use pipes and filters in the same way (if the have the same interface)
- abstract implementation of read-, write- and run-scenario (active-filters)
  If all possible scenarios are implemented in an abstract class, it's very easy to write new filters. You don't have to write the whole scenarios again and again.
- using generics for datatype-transparency
  Your pipes/filters will work with every datatype if you use generics

1) work with interfaces

Filters are only referencing Interfaces for Predecessors and Successors. These interfaces are implemented by the pipes AND the filters. Theres no difference for the filter, whether it uses a pipe or a filter.



2) abstract implementation of read-, write- and run-scenario (active-filters)

If you analize the certain scenarios of the different filters, you will see that the algorithm looks all the time the same. There are just some differences in the filter-specific parts. For this parts we use abstract methods which have to be implemented by the certain concrete filters.

e.g.: PULL-strategy – WordFilter

```
create a new Word W
read next Character C
do until C == " "
    append C to W

return W
```

e.g.: PULL-strategy – LineFilter

```
create a new Line L
read next Word W
do until L is full
    append W to L

return L
```

The differences between the PULL-scenario of the WordFilter and the LineFilter are:

1. different datatypes

   --> *use Generics*

2. different loop-conditions

   --> *use abstract method*

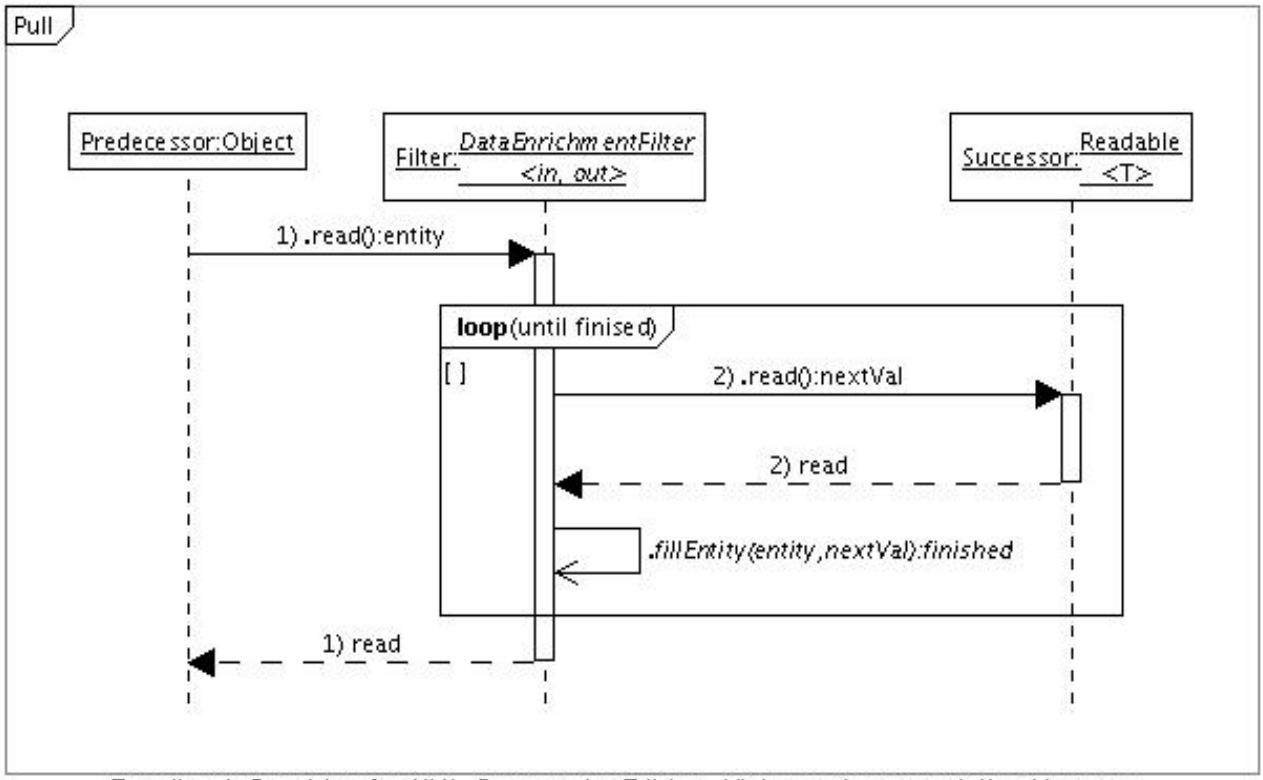
3. different appending-methods

   --> *use abstract method*

Abstract implementation of the read-scenario (PULL)

note: getNewEntity(), and fillEntity(entity, input) are abstract methods

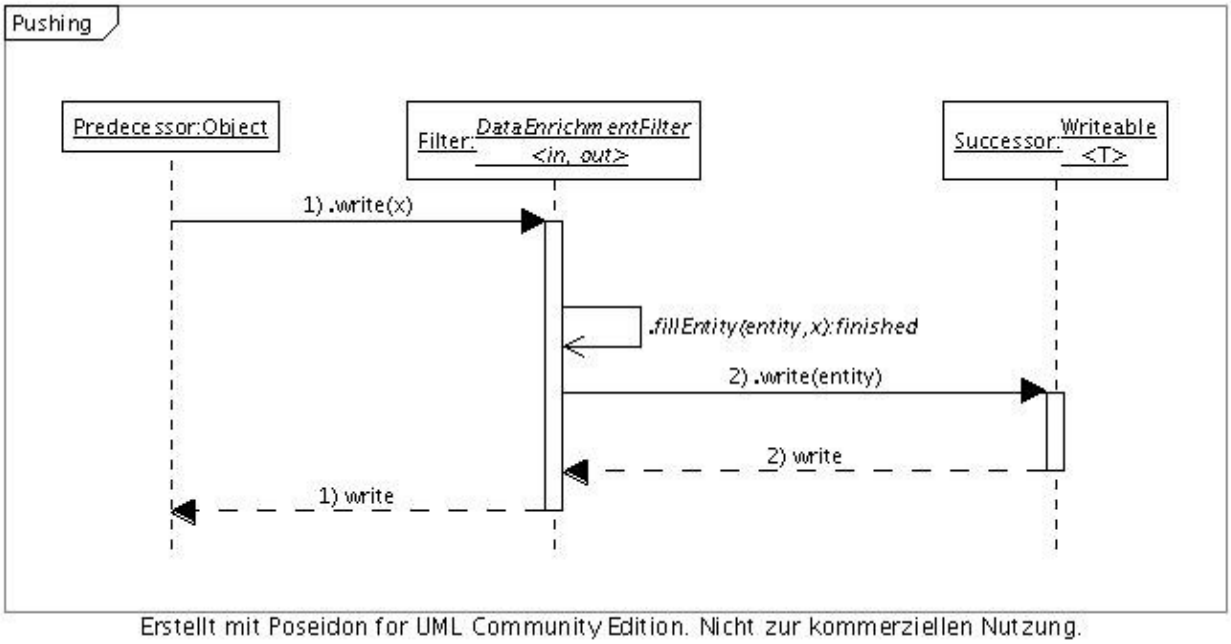
```
read()
    entity = getNewEntity()
    do
        input = readInput();
        finished = fillEntity(entity, input)
    until finished == true
    return entity;
```



Abstract implementation of the write-scenario (PULL)

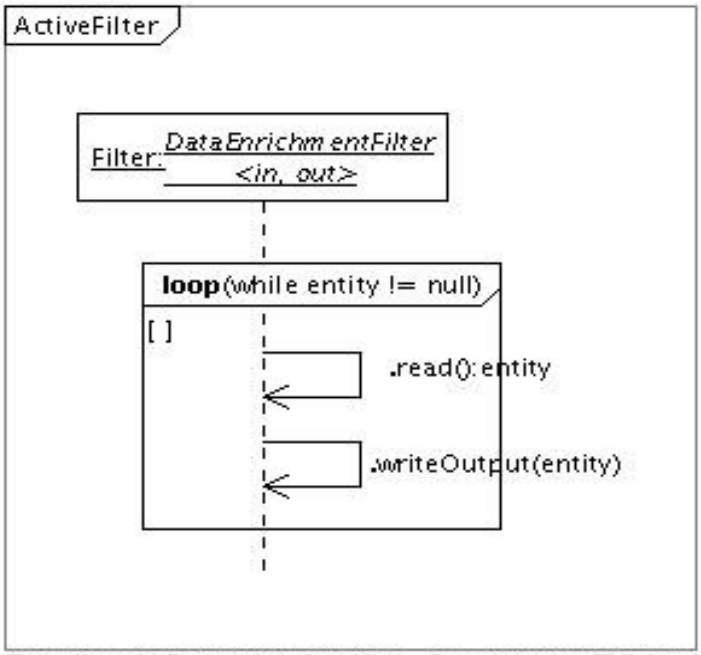
note: getNewEntity(), and fillEntity(entity, input) are abstract methods

```
write(nextValue)
        if (entity == null)
                entity = getNewEntity()
        finished = fillEntity(entity, nextValue);
        if (finished = true)
                successor.write(entity);
```



Abstract implementation of the run-scenario (active-filter)

```
run()
        while streamIsAlive
                entity = read();
                successor.write(entity);
        loop
```



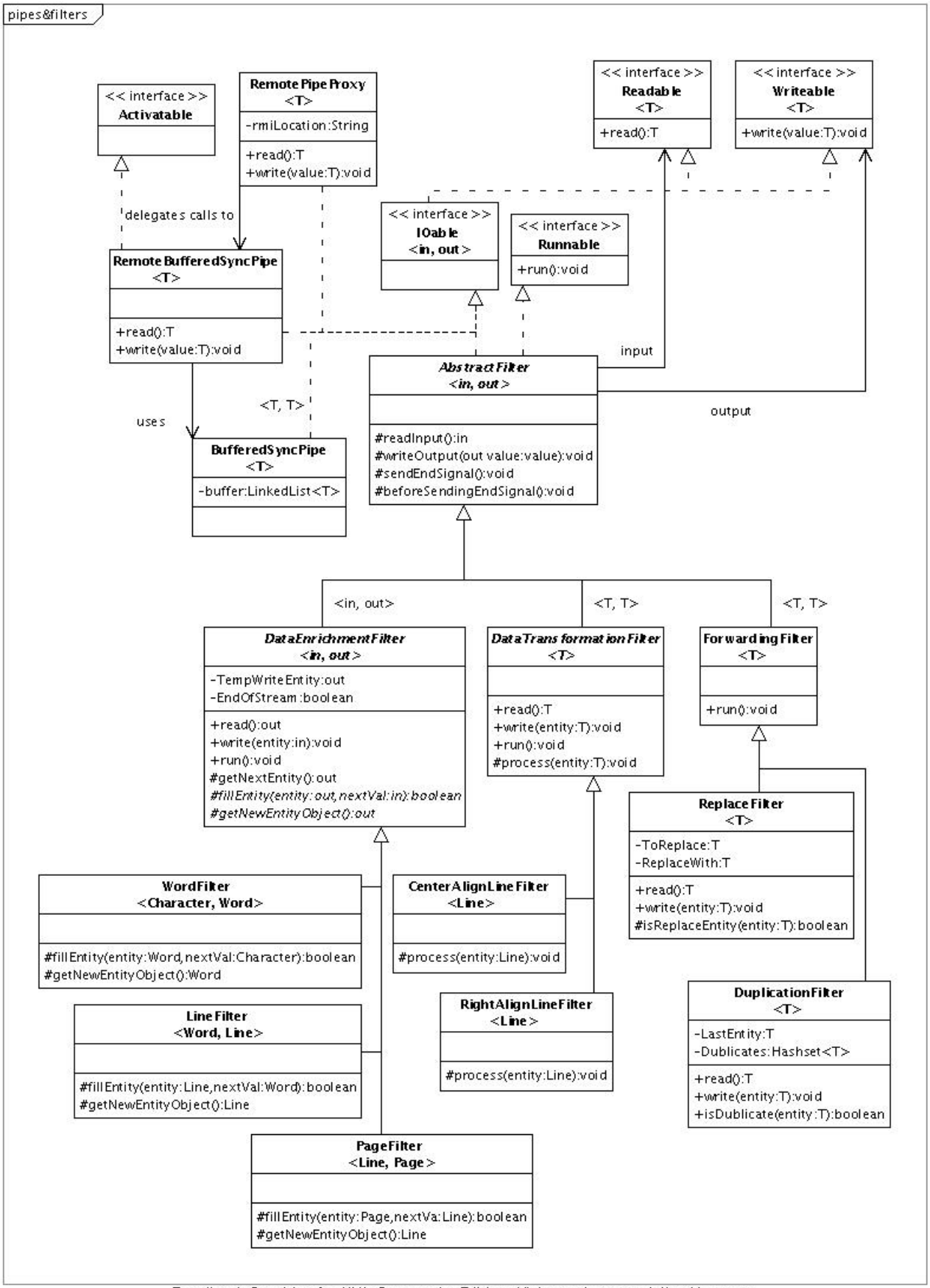
For implementing a concrete filter, only 2 methods have to be implemented.

e.g.: WordFilter

```
protected Word getNewEntityObject() {
    return new Word();
}


protected boolean fillEntity(Character nextVal, Word entity) {
    if (nextVal != null && !nextVal.equals(DELIMITER)) {
        entity.appendChar(nextVal);
        return false;
    }
    return true;
}
```

The whole architecture



Different types of filters

DataEnrichmentFilter

Filters wich have different datatypes for input and output. They enrich the data they get as input.

e.g.: The word Filter builds up Lines from several words.

DataTransformationFilter

Filters wich only transform the input-data but don't change the DataType.

e.g.: CenterAlignLineFilter

ForwardingFilter

Filters which do not transform the data, but only forward some special parts of the input-data.

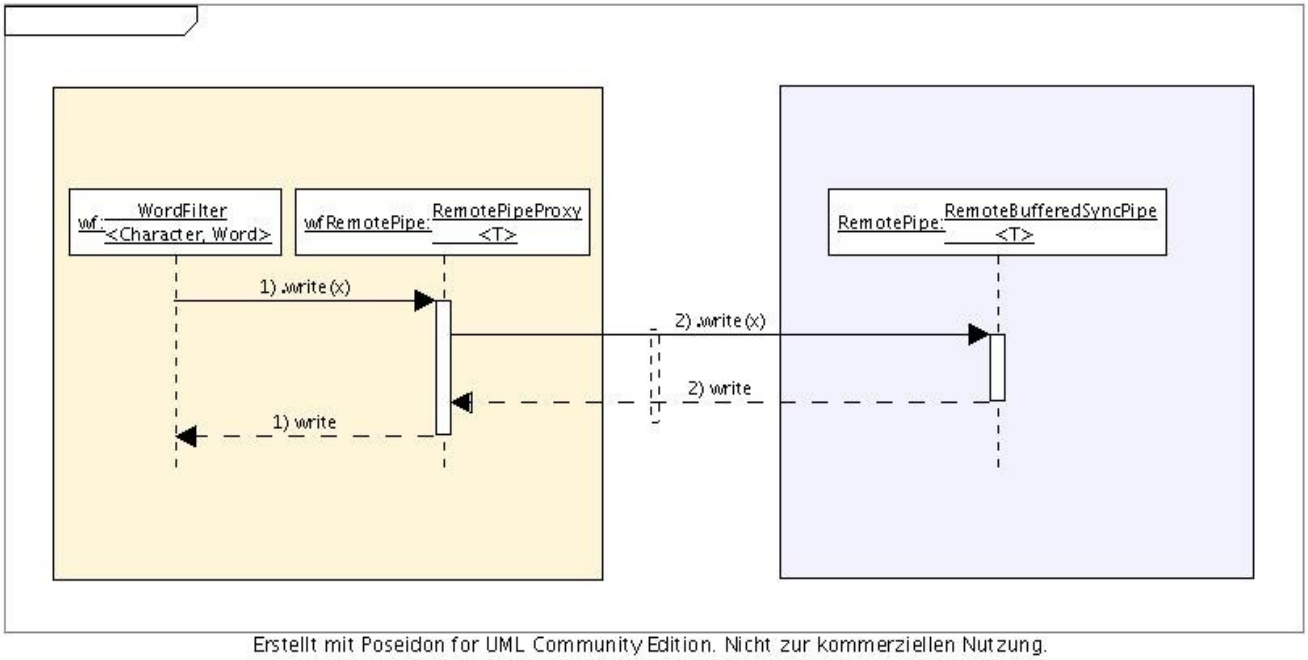
e.g.: DublicationFilter (Filters double occuring BLANKs).

Distributed Pipe

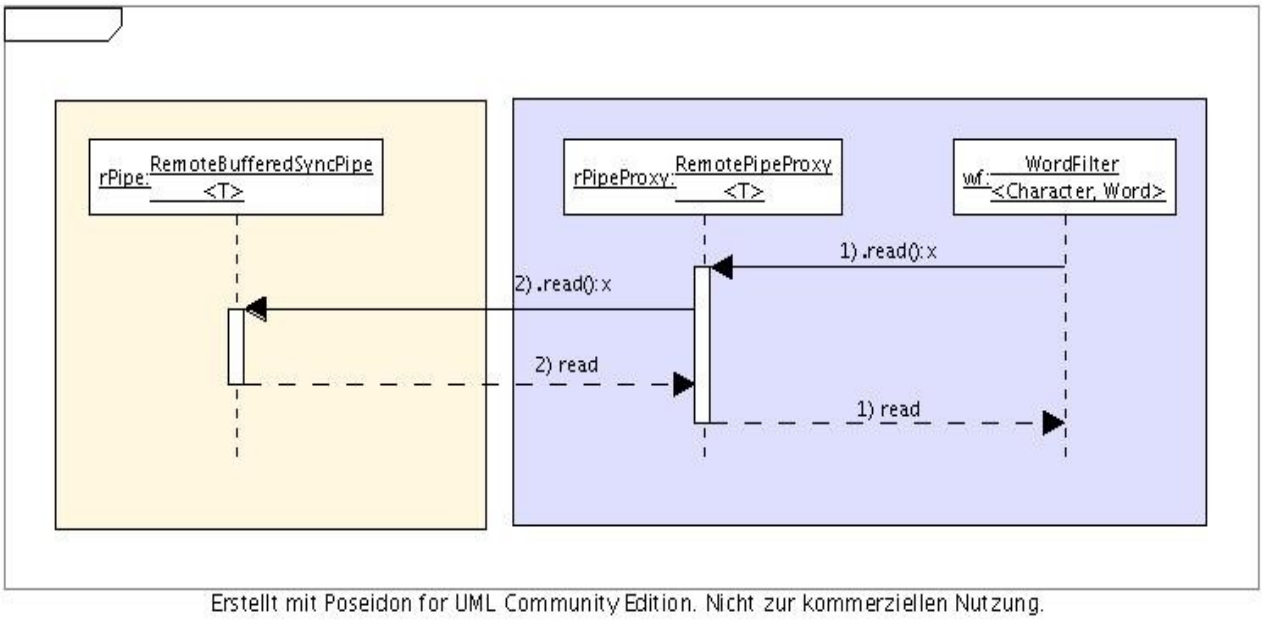
For the distributed Pipe we decided to split up our pipe. The local processes use a PipeProxy for calling a remote Pipe. The PipeProxy handles the rmi-stuff like looking up and handling the rmi-exceptions.

The RemotePipe just decorates the BufferedSyncPipe, so no additional implementation is needed.

RemotePipe - Pushing



RemotePipe – Pulling



Performance

tested with alice30.txt (163.218 characters):

local execution

multithreaded execution	~ 1 sec
push/pull-strategy	< 1 sec

distributed execution

(half of the pipe on Computer1, half of the pipe on Computer2)

using WirlessLAN:	> 1 minute
using LAN:	~ 29 sec

--> transmission needs to long, local processing is much faster!