Service to process documents with Alpino on Lisa

Paul Huygen <paul.huygen@huygen.nl>

10th March 2013

Abstract

This document creates and documents a service to process text-files with the Alpino dependency-parser on the Lisa supercomputer.

Contents

1 *Introduction* 2
1.1 How does it work? 2
1.1.1 Trays 2
1.1.2 Process manager 2
1.1.3 Parse jobs 2
1.1.4 Synchronisation 2
2 *Implementation* 3
2.1 Directory structure 3
2.2 Parser 3
2.3 Synchronisation mechanism 4
2.4 Temporary files 6
2.5 Log mechanism 6
2.6 Manage the jobs 7
2.6.1 Submit extra jobs 10
2.7 The parse job 11
2.7.1 Notification 17
2.8 Process manager 18
2.8.1 When will the manager run 18
2.8.2 Frequent tasks and less frequent tasks 18
2.9 Upload script 21
2.10 State script 22
2.11 Download script 22
2.12 Download a directory with scripts 23
A *Translate and run* 26
A.1 Pre-processing 27
A.1.1 Process dollar characters 27
A.2 Typeset this document 27
A.2.1 The w2pdf script 28
A.3 Install the service 30
A.4 create the program sources 31
B *Indexes* 31
C *Filenames* 31
D *Macro's* 32
E *Variables* 33

1 Introduction

- Need for Natural Language Processing (NLP) tools that are resource-intensive.
- Unleash the power of our supercomputer for this.
- Have nodes of the supercomputer to process documents in parallel with the same NLP parser.
- Scale advantage.
- Users from the University campus copy their documents into an in-tray and find after some time the processed version of the document in an out-tray.

1.1 How does it work?

1.1.1 Trays

- In-tray collects files that have been submitted by users.
- When a parser processes a file it moves the file to a process-tray.
- When the parser finishes a parse it writes the result as a file in an out-tray and removes the file from the process-tray.
- When the processing of the file takes too long, the file is moved from the processing-tray to a time-out tray.
- If the parser has been killed before it has finished its job, the file is replaced from the process-tray into the in-tray.
- Users collect the results from the out-tray.

1.1.2 Process manager

- Starts periodically (cron job).
- Checks whether users have submitted files.
- Submits a sufficient amount of parser jobs that process the input-files.
- Checks whether parser jobs have been killed and moves unprocessed files from the process-tray back into the in-tray.
- Checks synchronisation mechanism (cf. section 2.3)

1.1.3 Parse jobs

- Submitted by the process manager.
- Each job runs on a single node of the supercomputer.
- Use the cores of the node to process multiple documents simultaneously.
- How to pass arguments for the parser? Two possibilities: 1) in an extra file with a name that is related to the first file; 2) on the first line of the input file.

1.1.4 Synchronisation

- Prevent that two processes select the same input file for processing.
- File selection must be atomic action.
- Creation of a directory is atomic action, therefore generate a semaphore mechanism that works by creating and removing a block-directory.
- Problem: Possibility that a process is killed before it would remove the block-directory that it has created.
- Solution: Process-manager kills old block-directories.
- Problem: Possibility that process copies an input file that is still in status nascendi.
- Two possible solutions: 1) user submits the file and then second file with e similar name. If the second file is present, the copying of the first file must have been completed; 2) Process a file only when it is older than e.g. one minute.

2 Implementation

2.1 Directory structure

We have directories for the following:

trays: Each of the "trays" is a directory.
bin: A directory for binaries etc.

⟨ *parameters in Makefile* 3a ⟩ ≡
```
DIRS = /home/phuijgen/nlp/service/intray \
       /home/phuijgen/nlp/service/proctray \
       /home/phuijgen/nlp/service/outtray \
       /home/phuijgen/nlp/service/timeouttray \
       /home/phuijgen/nlp/service/bin
```
◇
Fragment defined by 3ac, 27a, 28c, 30c, 31c.
Fragment referenced in 26d.

⟨ *define variables for filenames* 3b ⟩ ≡
```
INBAK=/home/phuijgen/nlp/service/intray
UITBAK=/home/phuijgen/nlp/service/outtray
PROCBAK=/home/phuijgen/nlp/service/proctray
TOOOBAK=/home/phuijgen/nlp/service/timeouttray
PROJROOT=/home/phuijgen/nlp/service
cd $PROJROOT
```
◇
Fragment defined by 3b, 7b, 13e.
Fragment referenced in 12b, 18b, 21ah, 22b, 23a.
Defines: `INBAK` 11ag, 15cd, 17g, 19l, 20a, 21a, 22a, `PROCBAK` 14a, 15d, 17ai, 19l, 20a, `UITBAK` 17acdk, 20d, 23a.

2.2 Parser

This document implements a service for the Alpino parser (http://www.let.rug.nl/vannoord/alp/Alpino/). So we obviously need this thing.

⟨ *parameters in Makefile* 3c ⟩ ≡
```
ALPINO_TARBALL = Alpino-x86_64-linux-glibc2.5-19744.tar.gz
ALPINO_URL = http://www.let.rug.nl/vannoord/alp/Alpino/binary/versions/Alpino-x86_64-linux-glibc2.5-19
LOCAL_ALPINO = /home/phuijgen/nlp/service/bin/Alpino-x86_64-linux-glibc2.5-19744.tar.gz
```
◇
Fragment defined by 3ac, 27a, 28c, 30c, 31c.
Fragment referenced in 26d.

⟨ *expliciete make regels* 3d ⟩ ≡
```
$(LOCAL_ALPINO) :
        cd /home/phuijgen/nlp/service/bin && wget http://www.let.rug.nl/vannoord/alp/Alpino/binary/ver
```
◇
Fragment defined by 3d, 27bc, 28d, 31d.
Fragment referenced in 26d.

2.3 Synchronisation mechanism

Make a mechanism that ensures that only a single process can execute some functions at a time. For instance, if a process selects a file to be processed next, it selects a file name from a directory-listing and then removes the selected file from the directory. The two steps form a "critical code section" and only a single process at a time should be allowed to execute this section. Therefore, generate the functions `passeer` and `veilig` (cf. E.W. Dijkstra). When a process completes `passeer`, no other processes can complete `passeer` until the first process executes `veilig`.

Function `passeer` tries repeatedly to create a *lock directory*, until it succeeds and function `veilig` removes the lock directory.

Sometimes de-synchonisation is good, to prevent that all processes are waiting at the same time for the same event. Therefore, now and then a process should wait a random amount of time. We don't need to use sleep, because the cores have no other work to do.

⟨ *synchronisation functions* 4a ⟩ ≡

```
waitabit()
{ ( RR=$RANDOM
    while
      [ $RR -gt 0 ]
    do
    RR=$((RR - 1))
    done
  )

}

◇
```

Fragment defined by 4ab, 5a.
Fragment referenced in 12b, 18b, 21ah, 22b.

⟨ *synchronisation functions* 4b ⟩ ≡

```
export LOCKDIR=/home/phuijgen/nlp/service/lock

function passeer () {
 while ! (mkdir $LOCKDIR 2> /dev/null)
 do
   waitabit
 done
}

function veilig () {
  rmdir "$LOCKDIR"
}

◇
```

Fragment defined by 4ab, 5a.
Fragment referenced in 12b, 18b, 21ah, 22b.
Defines: `LOCKDIR` 5a, `passeer` 5bc, 6a, 7c, 8ab, 9b, 10a, `veilig` 5bc, 6a, 7c, 8ab, 9b, 10a.

The processes that execute these functions can crash and they are killed when the time alotted to them has been used up. Thus it is possible that a process that executed `passeer` is not able to execute `veilig`. As a result, all other processes would come to a halt. Therefore, check the age of the lock directory periodically and remove the directory when it is older than, say, two minutes (executing critical code sections ought to take only a very short amount of time).

⟨ *synchronisation functions* 5a ⟩ ≡
```
find $LOCKDIR -amin +2 -print 2>/dev/null | xargs rm -rf
```
◇

Fragment defined by 4ab, 5a.
Fragment referenced in 12b, 18b, 21ah, 22b.
Uses: LOCKDIR 4b, print 28a.

The synchronisation mechanism can be used to have parallel processed update the same counter.

⟨ *increment filecontent* 5b ⟩ ≡
```
passeer
NUM=‘cat @1‘
echo $((NUM + 1 )) > @1
veilig
```
◇

Fragment referenced in 13b.
Uses: passeer 4b, veilig 4b.

⟨ *decrement filecontent* 5c ⟩ ≡
```
passeer
NUM=‘cat @1‘
echo $((NUM - 1 )) > @1
veilig
```
◇

Fragment referenced in 13b.
Uses: passeer 4b, veilig 4b.

We will need a mechanism to find out whether a certain operation has taken place within a certain past time period. We use the timestamp of a file for that. When the operation to be monitored is executed, the file is touched. The following macro checks such a file. It has the following three arguments: 1) filename; 2) time-out period; 3) result. The result parameter will become true when the file didn't exist or when it had not been touched during the time-out period. In those cases the macro touches the file.

⟨ *check whether update is necessary* 6a ⟩ ≡
```
⟨ write log (6b now: `date +%s` ) 7a ⟩
arg=@1
stamp=`date -r @1 +%s`
⟨ write log (6c $arg: $stamp ) 7a ⟩
passeer
if [ ! -e @1 ]
then
  @3=true
elif [ $((`date +%s` - `date -r @1 +%s`)) -gt @2 ]
then
  @3=true
else
  @3=false
fi
if $@3
then
  echo `date` > @1
fi
veilig
if $@3
then
  ⟨ write log (6d yes, update ) 7a ⟩
else
  ⟨ write log (6e no, no update ) 7a ⟩
fi
◇
```
Fragment referenced in 19bg.

2.4 Temporary files

We will often use temporary files. Generate a filename for a temporary file that will be removed after use.

⟨ *create name for tempfile* 6f ⟩ ≡
```
tmpfil=`mktemp --tmpdir tmp.XXXXXXX`
rm -rf $tmpfil
◇
```
Fragment referenced in 12b, 18b, 21ah, 23a.
Defines: `tmpfil` 8ab, 9ab, 10a, 21a.
Uses: `tmpdir` 24c.

2.5 Log mechanism

Write to a log file if logging is set to true.

⟨ *init logfile* 6g ⟩ ≡
```
LOGGING=true
LOGFIL=/home/phuijgen/nlp/service/logfil
PROGNAM=@1
◇
```
Fragment referenced in 12b, 18b, 21ah, 22b, 23ac.
Defines: `LOGFIL` 7a, `LOGGING` 7a.

⟨ *write log* 7a ⟩ ≡
```
if LOGGING=true
then
  echo `date`";" $PROGNAM":" @1 >>$LOGFIL
fi
```
◇

Fragment referenced in 6a, 7c, 11ag, 12b, 16a, 17ae, 18b, 19bg, 21ah, 22b, 23c, 24a, 25a, 26a.
Uses: LOGFIL 6g, LOGGING 6g.

2.6 Manage the jobs

When we have received files to be parsed we have to submit the proper amount of parse jobs. To determine whether new jobs have to be submitted we have to know the number of waiting and running jobs. Unfortunately it is too costly to often request a list of running jobs. Therefore we will make a bookkeeping. File `/home/phuijgen/nlp/service/.jobcount` contains a list of the running and waiting jobs.

⟨ *define variables for filenames* 7b ⟩ ≡
```
JOBCOUNTFILE=/home/phuijgen/nlp/service/.jobcount
```
◇

Fragment defined by 3b, 7b, 13e.
Fragment referenced in 12b, 18b, 21ah, 22b, 23a.
Defines: JOBCOUNTFILE 7c, 8ab, 9b, 10a, 11a, 20c.

It is updated as follows:

- When a job is submitted, a line containing the job-id, the word “wait” and a timestamp is added to the file.
- A job that starts, replaces in the line with its job-id the word “waiting” by running and replaces the timestamp.
- A job that ends regularly removes the line with its job-id.
- A job that ends leaves a log message. The filename consists of a concatenation of the jobname, a dot, the character “o” and the job-id. At a regular basis the existence of such files is checked and `$JOBCOUNTFILE` updated.

Submit a job and write a line in the jobcountfile. The line consists of the jobnumber, the word “wait” and the timestamp in universal seconds.

⟨ *submit a job* 7c ⟩ ≡
```
passeer
qsub /home/phuijgen/nlp/service/bin/alpi | \
 gawk -F"." -v tst=`date +%s`  '{print $1 " wait " tst}' \
 >> $JOBCOUNTFILE
⟨ write log (7d Updated jobcountfile ) 7a ⟩
veilig
```
◇

Fragment referenced in 11e.

When a job starts, replace "wait" by "run". First find out what the job number is. The job ID begins with the number, e.g. 6670732.batch1.irc.sara.nl . Note the unexpected pattern in the Bash string replacement instruction.

⟨ *find out the job number* 7e ⟩ ≡
```
JOBNUM=${PBS_JOBID%%.*}
```
◇

Fragment referenced in 12b.

⟨ *change "wait" to "run" in jobcountfile* 8a ⟩ ≡

```
if [ -e $JOBCOUNTFILE ]
then
  passeer
  mv $JOBCOUNTFILE $tmpfil
  gawk -v jid=$JOBNUM -v stmp=‘date +%s‘ \
    ’{ if(match($0,"^"jid)>0) {print jid " run  " stmp} else {print}}’ \
    $tmpfil >$JOBCOUNTFILE
  veilig
  rm -rf $tmpfil
fi
◇
```

Fragment referenced in 12b.
Uses: JOBCOUNTFILE 7b, passeer 4b, print 28a, tmpfil 6f, veilig 4b.

When a job ends, it removes the line:

⟨ *remove the job from the counter* 8b ⟩ ≡

```
passeer
mv $JOBCOUNTFILE $tmpfil
gawk -v jid=$JOBNUM  ’$1 !~ "^"jid {print}’ $tmpfil >$JOBCOUNTFILE
veilig
rm -rf $tmpfil
◇
```

Fragment referenced in 12b.
Uses: JOBCOUNTFILE 7b, passeer 4b, print 28a, tmpfil 6f, veilig 4b.

Periodically check whether jobs have been killed before completion and have thus not been able to remove their line in the jobcountfile. To do this, write the jobnumbers in a temporary file and then check the jobcounter file in one blow, to prevent frequent locks.

⟨ *do brief check of expired jobs* 8c ⟩ ≡

```
obsfil=‘mktemp --tmpdir obs.XXXXXXX‘
rm -rf $obsfil
⟨ make a list of jobs that produced logfiles (8d $obsfil ) 9a ⟩
⟨ compare the logfile list with the jobcounter list (8e $obsfil ) 9b ⟩
rm -rf $obsfil
◇
```

Fragment referenced in 8f.

⟨ *do the frequent tasks* 8f ⟩ ≡

```
⟨ do brief check of expired jobs 8c ⟩
◇
```

Fragment defined by 8f, 20e.
Fragment referenced in 19g.

When a job has ended, a logfile, and sometimes an error-file, is produced. The name of the logfile is a concatenation of the jobname, a dot, the character `o` and the jobnumber. The error-file has a similar name, but the character `o` is replaced by `e`. Generate a sorted list of the jobnumbers and remover the logfiles and error-files:

⟨ *make a list of jobs that produced logfiles* 9a ⟩ ≡
```
for file in alpi.o*
do
  JOBNUM=${file##alpi.o}
  echo ${file##alpi.o} >>$tmpfil
  rm -rf alpi.[eo]$JOBNUM
done
sort < $tmpfil >@1
rm -rf $tmpfil
◇
```
Fragment referenced in 8c.
Uses: `tmpfil` 6f.

Remove the jobs in the list from the counter file if they occur there.

⟨ *compare the logfile list with the jobcounter list* 9b ⟩ ≡
```
if [ -e $JOBCOUNTFILE ]
then
  passeer
  sort < $JOBCOUNTFILE >$tmpfil
  gawk -v obsfil=@1 '
    BEGIN {getline obs < obsfil}
    { while((obs<$1) && ((getline obs < obsfil) >0)){}
      if(obs==$1) next;
      print
    }
  ' $tmpfil >$JOBCOUNTFILE
  veilig
fi
rm -rf $tmpfil
◇
```
Fragment referenced in 8c.
Uses: `JOBCOUNTFILE` 7b, `passeer` 4b, `print` 28a, `tmpfil` 6f, `veilig` 4b.

From time to time, check whether the jobs-bookkeeping is still correct. To this end, request a list of jobs from the operating system.

⟨ *verify jobs-bookkeeping* 9c ⟩ ≡
```
actjobs=‘mktemp --tmpdir act.XXXXXX‘
rm -rf $actjobs
qstat -u  phuijgen | grep alpi | gawk -F"." '{print $1}' \
 | sort  >$actjobs
⟨ compare the active-jobs list with the jobcounter list (9d $actjobs ) 10a ⟩
rm -rf $actjobs
◇
```
Fragment referenced in 9e.

⟨ *do the now-and-then tasks* 9e ⟩ ≡
```
⟨ verify jobs-bookkeeping 9c ⟩
◇
```
Fragment defined by 9e, 20f.
Fragment referenced in 19b.

⟨ *compare the active-jobs list with the jobcounter list* 10a ⟩ ≡

```
if [ -e $JOBCOUNTFILE ]
then
  passeer
  sort < $JOBCOUNTFILE >$tmpfil
  gawk -v actfil=@1 -v stmp='date +%s' '
    ⟨ awk script to compare the active-jobs list with the jobcounter list 10b ⟩
  ' $tmpfil >$JOBCOUNTFILE
  veilig
  rm -rf $tmpfil
else
  cp @1 $JOBCOUNTFILE
fi
◇
```

Fragment referenced in 9c.
Uses: JOBCOUNTFILE 7b, passeer 4b, tmpfil 6f, veilig 4b.

Copy lines from the logcount file if the jobnumber matches a line in the list actual jobs. Write entries for jobnumbers that occur only in the actual job list.

⟨ *awk script to compare the active-jobs list with the jobcounter list* 10b ⟩ ≡

```
BEGIN {actlin=(getline act < actfil)}
{ while(actlin>0 && (act<$1)){
     print act " wait " stmp;
     actlin=(getline act < actfil);
  };
  if((actlin>0) && act==$1 ){
     print
     actlin=(getline act < actfil);
  }
}
END {
    while((actlin>0) && (act ~ /^[[:digit:]]+/)){
      print act " wait " stmp;
    actlin=(getline act < actfil);
 };
}
◇
```

Fragment referenced in 10a.
Uses: print 28a.

2.6.1 Submit extra jobs

Check how many files have to be parsed (NRFILES) and how many jobs there are (NRJOBS). If there are more than 50 files per job, submit extra jobs.

When before submitting jobs it turns out that, although no job is running at all, there are files in proctray. In that case, they can be moved back to the intray.

⟨ *check/perform every time* 10c ⟩ ≡

```
⟨ replace files from proctray when no processes are running 20a ⟩
⟨ submit jobs when necessary 11a ⟩
◇
```

Fragment referenced in 19a.

⟨ *submit jobs when necessary* 11a ⟩ ≡
```
NRFILES=‘ls -1 $INBAK |  wc -l‘
if [ -e $JOBCOUNTFILE ]
then
  NRJOBS=‘wc -l < $JOBCOUNTFILE‘
else
  NRJOBS=0
fi
```
⟨ *derive number of jobs to be submitted* (11b `SUBJOBS`) 11f ⟩
⟨ *write log* (11c `start $SUBJOBS jobs`) 7a ⟩
⟨ *submit extra jobs* (11d `SUBJOBS`) 11e ⟩
◇

Fragment referenced in 10c.

⟨ *submit extra jobs* 11e ⟩ ≡
```
for ((a=1; a <= @1;  a++))
do
```
  ⟨ *submit a job* 7c ⟩
```
done
```
◇

Fragment referenced in 11a.

⟨ *derive number of jobs to be submitted* 11f ⟩ ≡
```
REQJOBS=$(( $(( $NRFILES / 50 )) ))
if [ $NRFILES -gt 0 ]
then
  if [ $REQJOBS -eq 0 ]
  then
    REQJOBS=1
  fi
fi
@1=$(( $REQJOBS - $NRJOBS ))

```
◇

Fragment referenced in 11a.

2.7 The parse job

Now let us generate the code for the jobs that can be submitted by the process manager and that perform the actual parsing work. Currently a job requests only a single node and it runs for half an hour maximum.

A job checks whether there is something to do, otherwise it stops.

⟨ *do not run when the intray is empty* 11g ⟩ ≡
```
NRFILES=‘ls -1 $INBAK |  wc -l‘
if [ $NRFILES -eq 0 ]
then
```
  ⟨ *write log* (11h `Nothing to do. Quit.`) 7a ⟩
```
  exit
fi
```
◇

Fragment referenced in 12b.

⟨ *PBS job parameters* 12a ⟩ ≡
```
#PBS -lnodes=1
#PBS -lwalltime=30:00
◇
```
Fragment referenced in 12b.

"../bin/alpi" 12b≡
```
⟨ PBS job parameters 12a ⟩
STARTTIME=‘date +%s‘
⟨ init logfile (12c alpi ) 6g ⟩
⟨ synchronisation functions 4a, ... ⟩
⟨ define variables for filenames 3b, ... ⟩
⟨ do not run when the intray is empty 11g ⟩
⟨ create name for tempfile 6f ⟩
⟨ find out the job number 7e ⟩
PROGNAM=alpi$JOBNUM
⟨ write log (12d start ) 7a ⟩
⟨ change “wait” to “run” in jobcountfile 8a ⟩
⟨ load sara modules 12f ⟩
⟨ functions in alpars 15a, ... ⟩
⟨ set environments for the NLP applications 12g, ... ⟩
cd $TMPDIR
⟨ start parallel processes that perform the parsing 13b ⟩
wait
⟨ remove the job from the counter 8b ⟩
⟨ write log (12e stop ) 7a ⟩
exit
◇
```

Sara developed modules to facilitate job tasks. We will use module `disparm` [1], to retrieve the number of cores that are available.

⟨ *load sara modules* 12f ⟩ ≡
```
module load disparm
◇
```
Fragment referenced in 12b.
Defines: `disparm` Never used.

Set environment variables for the NLP applications. Currently only Alpino is a known and supported application.

Set the variables `ALPINO_HOME` and `PATH`.

⟨ *set environments for the NLP applications* 12g ⟩ ≡
```
export ALPINO_HOME=$HOME/nlp/Alpino
export PATH=$PATH:$ALPINO_HOME/bin
◇
```
Fragment defined by 12g, 13a.
Fragment referenced in 12b.

Set variables to configure Alpino. Gert-Jan van Noort wrote in a personal communication that the following two parameters must be set in order to have Alpino produce correct UTF:

1. https://www.sara.nl/systems/lisa/software/disparm

⟨ *set environments for the NLP applications* 13a ⟩ ≡
```
export SP_CSETLEN=212
export SP_CTYPE=utf8
◇
```
Fragment defined by 12g, 13a.
Fragment referenced in 12b.
Defines: SP_CSETLEN Never used, SP_CTYPE Never used.

Find out how many cores there are and then create the same number of parallel processes. The number of cores can be obtained from variable `sara-get-num-cores` that is supplied by the `disparm` module. Generate a file `proctal` that contains the number of processes that actually runs.

⟨ *start parallel processes that perform the parsing* 13b ⟩ ≡
```
echo 0 >$TMPDIR/proctal
export NCORES=‘sara-get-num-cores‘
echo "Node has " $NCORES " cores".
PROCNUM=0
for ((i=1; i<= NCORES ; i++)) ; do
  ( echo Process number: $PROCNUM
    ⟨ increment filecontent (13c $TMPDIR/proctal ) 5b ⟩
    echo $PROCNUM: Proctal after increment: ‘cat $TMPDIR/proctal‘
    ⟨ perform a single process 14a ⟩
    ⟨ decrement filecontent (13d $TMPDIR/proctal ) 5c ⟩
    echo $PROCNUM: Proctal after decrement: ‘cat $TMPDIR/proctal‘
  ) &
  PROCNUM=$(($PROCNUM + 1))
done
◇
```
Fragment referenced in 12b.

Each process repeats the following:

1. Select a file from the in-tray
2. Move it to the process-tray
3. Copy it to a temporary file on the local filesystem
4. Run Alpino on it.
5. Copy the result to the out-tray.
6. Send the result to the user.
7. Remove the source from the process-tray

This code is a bit demo-style. When the input-file is moved into the process-tray it should get a name that links it to this process, to facilitate moving it back to the intray when this process dies prematurely.

The function `getfile` stores a filename in variable `FILNAM` and moves the input-file with this name to the process-tray. Its result reflects whether it actually found a file.

To keep track of the time needed to perform the parsing, the processing time will be measured and recorded in a bookkeepfile.

⟨ *define variables for filenames* 13e ⟩ ≡
```
export BOOKKEEPFILE=/home/phuijgen/nlp/service/processed_files
◇
```
Fragment defined by 3b, 7b, 13e.
Fragment referenced in 12b, 18b, 21ah, 22b, 23a.
Defines: BOOKKEEPFILE 14f.

Note that, to use the time command with its arguments, the "time" command must be given as the full path to the binary (see https://bugs.launchpad.net/ubuntu/+source/time/+bug/220512).

⟨ *perform a single process* 14a ⟩ ≡

```
while getfile
do
  BTIME=‘date +%s‘
  echo Got file $FILNAM
  INFIL=‘mktemp --tmpdir‘
  OUTFIL=‘mktemp --tmpdir‘
  rm -rf $INFIL
  rm -rf $OUTFIL
  cp $PROCBAK/$FILNAM $INFIL
  ⟨ generate format for time command (14b TIMEFORMAT ) 14g ⟩
  ⟨ construct the alpino command (14c $INFIL ) 16c ⟩
  ⟨ apply the alpino command (14d $INFIL,14e $OUTFIL ) 16a ⟩
  ⟨ send the output to the sender 17a ⟩
  echo $FILNAM processed
  ETIME=‘date +%s‘
  ⟨ log time lapse 14f ⟩
done
```

◇

Fragment referenced in 13b.

Make a list of the files that have been processed, relevant properties and the wall-times that they needed. The list has the following columns: 1) filesize; 2) wall time; 3) filename and 4) processing command.

⟨ *log time lapse* 14f ⟩ ≡

```
echo ‘stat --printf="%s" $INFIL‘"       $(($ETIME - $BTIME))    $FILNAM $ALPCOMMAND" >> $BOOKKEEPFILE
```

◇

Fragment referenced in 14a.
Uses: BOOKKEEPFILE 13e.

⟨ *generate format for time command* 14g ⟩ ≡

```
@1=‘stat --printf="%s" $INFIL‘"\t%e\t$FILNAM\t$ALPCOMMAND"
```

◇

Fragment referenced in 14a.

The function `getfile` does the following: When there are files in the input-tray it selects the name of the largest file, stores the name in variable `FILNAM` and moves the file into the process-tray. When the input-tray is empty, the function sleeps a while and then retries.

When this is successful, it returns value 0, otherwise

⟨ *functions in alpars* 15a ⟩ ≡
```
filefind()
{
  while
    ⟨ fetch a file from the intray (15b FILNAM ) 15c ⟩
    [ -z "$FILNAM" ]
  do
    waitabit
  done
}
◇
```
Fragment defined by 15ad.
Fragment referenced in 12b.

Fetch a file from the intray by moving it to the outtray. File movement ought to be an atomic operation, so two processes cannot move the same file at the same time. When less than half of the time alotted to this job has been expired, fetch the largest file from the intray. Otherwise, fetch the smallest file.

⟨ *fetch a file from the intray* 15c ⟩ ≡
```
NOW=‘date +%s‘
if [ $((NOW - STARTTIME)) -lt 900 ]
then
  @1=‘ls -1S $INBAK 2>/dev/null | head -n 1‘
else
  @1=‘ls -1rS $INBAK 2>/dev/null | head -n 1‘
fi
◇
```
Fragment referenced in 15a.
Uses: INBAK 3b.

⟨ *functions in alpars* 15d ⟩ ≡
```
function getfile () {
  while filefind
  do
    if ( set -o noclobber; cat $INBAK/$FILNAM > $PROCBAK/$FILNAM ) 2> /dev/null;
    then
       waitabit
      break
    fi
  done
  rm -f $INBAK/$FILNAM
  touch $PROCBAK/$FILNAM
}

◇
```
Fragment defined by 15ad.
Fragment referenced in 12b.
Defines: getfile 14a.
Uses: INBAK 3b, PROCBAK 3b.

The job expects that the first line of input documents contains a she-bang, followed by the Alpino command to be applied. Hence, proceed as follows:

1. Extract the Alpino command and put it in variable AWCOMMAND.
2. Apply the Alpino command on the “tail” of the input file

3. Remove the temporary file.

⟨ *apply the alpino command* 16a ⟩ ≡
```
TIMEOUT=false
timeout 1710s bash -c "gawk 'NR>1' @1 | $ALPCOMMAND >@2 2>/dev/null"
if [ $? -eq 124 ]
then
  TIMEOUT=true
  ⟨ write log (16b $FILNAM time-out ) 7a ⟩
fi
◇
```
Fragment referenced in 14a.

⟨ *construct the alpino command* 16c ⟩ ≡
```
AWCOMMAND='NR==1 {gsub(/^#!/, "");print}'
ALPCOMMAND=`gawk "$AWCOMMAND" @1`
⟨ look for an xml-treebank request (16d ALPCOMMAND ) 16e ⟩
◇
```
Fragment referenced in 14a.

As far as I see, Alpino has a mode in which it produces a single output file, but there is also a mode in which it produces a directory with an xml-file for each sentence. To request a directory with xml-files, the user writes -XMLTREEBANK as option.

When the user has done that, generate a temporary directory with a directory to store the xml files. At the end of the job, we pack this directory in a tarball and return that.

In contrast with what the name of the macro suggests, the file will not be sent by-e-mail, because it seems that the mail system of Lisa is not capable to process a large amount of texts.

⟨ *look for an xml-treebank request* 16e ⟩ ≡
```
if [ `echo $ALPCOMMAND | grep -e "-XMLTREEBANK" | wc -l` -ge 1 ]
then
  XTMPDIR=`mktemp --tmpdir -d XML.XXXXXX`
  XMLDIR=$XTMPDIR/xml
  mkdir $XMLDIR
  echo Directory $XMLDIR
  AWCOMMAND='{gsub("-XMLTREEBANK", "-flag treebank '$XMLDIR' -flag end_hook xml");print}'
  echo $AWCOMMAND
  ALPCOMMAND=`echo $ALPCOMMAND | gawk "$AWCOMMAND"`
  echo $ALPCOMMAND
else
  XMLDIR=
fi
◇
```
Fragment referenced in 16c.
Uses: print 28a, tmpdir 24c.

If timeout has occurred, move the original file from the proctray to the timeout tray. Otherwise, remove that file and write the output of the parsing to a file in the outtray.

⟨ *send the output to the sender* 17a ⟩ ≡
```
if $TIMEOUT
then
  ⟨ write log (17b $FILNAM to time-out tray ) 7a ⟩
  mv $PROCBAK/$FILNAM $TOOOBAK
else
  if [ "$XTMPDIR" = "" ]
  then
    ⟨ write log (17c Send $OUTFIL to $UITBAK/$FILNAM ) 7a ⟩
     mv $OUTFIL $UITBAK/$FILNAM
  else
    echo Send xml-directory to $UITBAK/$FILNAM
    ⟨ pack/send the xml-directory and remove xml directory 17d ⟩
  fi
  rm $PROCBAK/$FILNAM
fi
◇
```
Fragment referenced in 14a.
Uses: UITBAK 3b.

Pack the directory with the XML files in a tarball, put the tarball in the outtray and then remove the temporary directory and the XTMPDIR variable.

⟨ *pack/send the xml-directory and remove xml directory* 17d ⟩ ≡
```
(
  cd $XTMPDIR
  tar -czf $FILNAM xml
  mv $FILNAM $UITBAK
)
rm -rf $XTMPDIR
XTMPDIR=
◇
```
Fragment referenced in 17a.
Uses: UITBAK 3b.

2.7.1 Notification

Report to the intermediate server what is going on, on a minutely basis.

⟨ *print notification* 17e ⟩ ≡
```
⟨ make list of tray (17f intray,17g $INBAK ) 17o ⟩
⟨ make list of tray (17h proctray,17i $PROCBAK ) 17o ⟩
⟨ make list of tray (17j outtray,17k $UITBAK ) 17o ⟩
⟨ make list of tray (17l toootray,17m $TOOOBAK ) 17o ⟩
⟨ report processors 18a ⟩
⟨ write log (17n notification sent ) 7a ⟩
◇
```
Fragment referenced in 22b.

⟨ *make list of tray* 17o ⟩ ≡
```
echo @1:
ls -1 @2
◇
```
Fragment referenced in 17e.

⟨ *report processors* 18a ⟩ ≡
```
echo processors: ‘grep run .jobcount | wc -l‘
```
◇
Fragment referenced in 17e.

2.8 Process manager

The process manager is a cron-job that is started periodically, but it is also started when a new file has been uploaded.

"../bin/alpinomanager" 18b≡
```
#!/bin/bash
⟨ init logfile (18c alpinomanager ) 6g ⟩
⟨ write log (18d start ) 7a ⟩
⟨ create name for tempfile 6f ⟩
⟨ synchronisation functions 4a, ... ⟩
⟨ stop if another alpinomanager is running 18g ⟩
⟨ define variables for filenames 3b, ... ⟩
⟨ perform the management tasks 19a ⟩
⟨ write log (18e stop ) 7a ⟩
```
◇

2.8.1 When will the manager run

The process manager could run as a cron-job in the background. However, we presume that this service will either be used intensively or not at all. Hence it makes sense that user requests start the alpinomanager at regular times.

⟨ *start the alpinomanager* 18f ⟩ ≡
```
( /home/phuijgen/nlp/service/bin/alpinomanager )
```
◇
Fragment referenced in 21ah, 22b.

Prevent that more than one instance of the program runs. I don’t know whether this is a really safe way. Note that the `ps` command gives a line for this alpinomanager and for the `ps` command as well.

⟨ *stop if another alpinomanager is running* 18g ⟩ ≡
```
COUNT=‘ps -A |grep alpinomanager | wc -l‘
if [ $COUNT -ge 3 ]
then
  exit
fi
```
◇
Fragment referenced in 18b.
Defines: `COUNT` Never used.

2.8.2 Frequent tasks and less frequent tasks

The jobmanager performs some tasks every time that it is started, some other tasks at intervals of about five minutes and yet other tasks every half hour. The time interval is controlled by the

date-stamps of indicator-files. When the file has been touched too long ago or when it does not exist, the tasks belonging to that file are performed and the file is touched.

⟨ *perform the management tasks* 19a ⟩ ≡
```
frequentcheckfile=/home/phuijgen/nlp/service/.briefjobcheck
now_and_then_checkfile=/home/phuijgen/nlp/service/.thoroughjobcheck
frequentcheckperiod=300
now_and_then_checkperiod=1800
⟨ check/perform now-and-then tasks 19b ⟩
⟨ check/perform frequent tasks 19g ⟩
⟨ check/perform every time 10c ⟩
◇
```
Fragment referenced in 18b.

⟨ *check/perform now-and-then tasks* 19b ⟩ ≡
```
⟨ check whether update is necessary (19c now_and_then_checkfile,19d $now_and_then_checkperiod,19e checkb ) 6a ⟩
if $checkb
then
⟨ write log (19f now-and-then tasks ) 7a ⟩
⟨ do the now-and-then tasks 9e, ... ⟩
fi
◇
```
Fragment referenced in 19a.

⟨ *check/perform frequent tasks* 19g ⟩ ≡
```
⟨ check whether update is necessary (19h frequentcheckfile,19i $frequentcheckperiod,19j checkb ) 6a ⟩
if $checkp
then
⟨ write log (19k frequent tasks ) 7a ⟩
⟨ do the frequent tasks 8f, ... ⟩
fi
◇
```
Fragment referenced in 19a.

Currently we assume that the parser is capable to process an input-file within half an our. Hence, files that have been longer than that time in the process-tray are assumed to be left-overs.

⟨ *restore old files from the process-tree* 19l ⟩ ≡
```
NRFILES=‘ls -1 $PROCBAK 2>/dev/null |  wc -l‘
if [ "$NRFILES" != "0" ]
then
  find $PROCBAK/* -amin +30 -print 2>/dev/null | xargs -Iaap mv aap  $INBAK
fi
◇
```
Fragment referenced in 20e.
Uses: INBAK 3b, print 28a, PROCBAK 3b.

Furthermore, when no processes are running, we can move all files from the proctray into the intray.

⟨ *replace files from proctray when no processes are running* 20a ⟩ ≡
```
⟨ determine number of running processes (20b RUNPROCCOUNT ) 20c ⟩
if [ "$RUNPROCCOUNT" == "0" ]
then
  NRFILES=‘ls -1 $PROCBAK 2>/dev/null |  wc -l‘
  if [ "$NRFILES" != "0" ]
  then
    find $PROCBAK/*  -print 2>/dev/null | xargs -Iaap mv aap  $INBAK
  fi
fi
◇
```
Fragment referenced in 10c.

⟨ *determine number of running processes* 20c ⟩ ≡
```
if [ -e $JOBCOUNTFILE ]
then
@1=‘gawk ’BEGIN {runprocs=0}; /run/ {runprocs++}; END {print runprocs}’ $JOBCOUNTFILE‘
else
 @1=0
fi
◇
```
Fragment referenced in 20a.
Uses: JOBCOUNTFILE 7b, print 28a.

Remove files that have been waiting too long before retrieval.

⟨ *remove old files from the outtray* 20d ⟩ ≡
```
NRFILES=‘ls -1 $UITBAK 2>/dev/null |  wc -l‘
if [ "$NRFILES" != "0" ]
then
  find $UITBAK/* -atime +1 -print | xargs rm -rf
fi
◇
```
Fragment referenced in 20f.
Uses: print 28a, UITBAK 3b.

⟨ *do the frequent tasks* 20e ⟩ ≡
```
⟨ restore old files from the process-tree 19l ⟩
◇
```
Fragment defined by 8f, 20e.
Fragment referenced in 19g.

⟨ *do the now-and-then tasks* 20f ⟩ ≡
```
⟨ remove old files from the outtray 20d ⟩
◇
```
Fragment defined by 9e, 20f.
Fragment referenced in 19b.

2.9 Upload script

An external computer can connect to Lisa by way of `ssh` and run the following script `download` to upload a file. The name of the file is given as argument and the contents is written to standard in.

To prevent that during the download a parse-process grabs the still incomplete file, the contents of the file is first collected into a temporary file. Then the file is moved to the intray (note that a move is an atomic action). Finally the alpinomanager is started.

```
"../bin/download" 21a≡
      #!/bin/bash
      # Download .. Download a file to be processed
      # Filename is argument
      # File contents from standard in
      ⟨ init logfile (21b download ) 6g ⟩
      ⟨ write log (21c start ) 7a ⟩
      ⟨ cd naar werkdir 21g ⟩
      ⟨ create name for tempfile 6f ⟩
      ⟨ synchronisation functions 4a, ... ⟩
      ⟨ define variables for filenames 3b, ... ⟩
      DESTNAME=$1
      DESTFILE=$INBAK/$DESTNAME
      cat > $tmpfil
      mv $tmpfil $DESTFILE
      ⟨ write log (21d start alpinomanager ) 7a ⟩
      ⟨ start the alpinomanager 18f ⟩
      ⟨ write log (21e alpinomanager finished ) 7a ⟩
      ⟨ write log (21f stop ) 7a ⟩
      exit
      ◇
```

```
⟨ cd naar werkdir 21g ⟩ ≡
      cd /home/phuijgen/nlp/service
      ◇
```

Fragment referenced in 21ah.

An alternative is, to upload a tar archive with files. This has the advantage that less SSH connections with Lisa are needed.

```
"../bin/download_archive" 21h≡
      #!/bin/bash
      # Download .. Download a tar archives with files to be processed
      # Filename is argument
      # File contents from standard in
      ⟨ init logfile (21i download_archive ) 6g ⟩
      ⟨ write log (21j start ) 7a ⟩
      ⟨ cd naar werkdir 21g ⟩
      ⟨ create name for tempfile 6f ⟩
      ⟨ synchronisation functions 4a, ... ⟩
      ⟨ define variables for filenames 3b, ... ⟩
      ⟨ unpack tar 22a ⟩
      ⟨ write log (21k start alpinomanager ) 7a ⟩
      ⟨ start the alpinomanager 18f ⟩
      ⟨ write log (21l stop ) 7a ⟩
      exit
      ◇
```

Unpack in a temporary directory to prevent that a job picks prematurely up a half-unpacked file.

⟨ *unpack tar* 22a ⟩ ≡
```
tmpdir=‘mktemp --tmpdir -d tmpd.XXXXXX‘
cd $tmpdir
tar -xzf -
mv * $INBAK
cd /home/phuijgen/nlp/service
rm -rf $tmpdir
◇
```
Fragment referenced in 21h.
Uses: INBAK 3b, tmpdir 24c.

2.10 State script

The external computer can connect to Lisa by way of `ssh` and ask for the status of a file that are to be processed. If that happens, Lisa send a list of the contents of the trays by e-mail and then returns the status of the file in the argument. The status can be 1) `waiting`; 2) `being processed`; 3) `ready` or 4) `unknown`.

"../bin/filstat" 22b≡
```
#!/bin/bash
# filstat: provide state of files
# Filename is argument
⟨ init logfile (22c filstat ) 6g ⟩
⟨ write log (22d start ) 7a ⟩
⟨ synchronisation functions 4a, ... ⟩
⟨ define variables for filenames 3b, ... ⟩
⟨ write log (22e start alpinomanager ) 7a ⟩
⟨ start the alpinomanager 18f ⟩
⟨ write log (22f alpinomanager ended ) 7a ⟩
⟨ print notification 17e ⟩
⟨ write log (22g stop ) 7a ⟩
◇
```

2.11 Download script

An external computer can connect to Lisa by way of `ssh` and run the following script `upload` to download a file. The name of the file is given as argument and the contents is written to standard out.

"../bin/upload" 23a≡
```
#!/bin/bash
# Upload .. upload a parse
# Filename is argument
⟨ init logfile (23b upload ) 6g ⟩
⟨ create name for tempfile 6f ⟩
⟨ define variables for filenames 3b, ... ⟩
FILNAM=$1
if [ -e $UITBAK/$FILNAM ]
then
 cat $UITBAK/$FILNAM
 rm -rf $UITBAK/$FILNAM
else
 echo "unknown: $UITBAK/$FILNAM"
fi
/home/phuijgen/nlp/service/bin/alpinomanager
◇
```

2.12 Download a directory with scripts

Instead of using standard services, a user can create a directory with software and use that software. The following script downloads and unpacks a directory. It gets the name of the directory root as argument and only if that is the correct name of the root directory in the tarball, the directory will be accepted.

When the script performed its task correctly it prints “OK”. Otherwise, it prints “error”.

"../bin/downloaddir" 23c≡
```
#!/bin/bash
# downloaddir -- Download a directory with software
# Directory name is argument
# Tarball from standard in
EXITSTATE=0
⟨ init logfile (23d downloaddir ) 6g ⟩
⟨ write log (23e start ) 7a ⟩
⟨ get the argument of the downloaddir script 24a ⟩
⟨ create a tempdir for downloaddir 24c ⟩
⟨ test and unpack the tarball 25a ⟩
⟨ remove the tempdir for downloaddir 24d ⟩
⟨ write log (23f stop ) 7a ⟩
⟨ echo exitstate 23g ⟩
exit $EXITSTATE
◇
```

⟨ echo exitstate 23g ⟩ ≡
```
if
  [ $EXITSTATE -eq 0 ]
then
  echo OK
else
  echo error
fi
◇
```
Fragment referenced in 23c.

⟨ *get the argument of the downloaddir script* 24a ⟩ ≡

```
DIRROOT=$1
if
 [ -z "$DIRROOT" ]
then
 echo error
 ⟨ write log (24b Stop: no argument ) 7a ⟩
 exit 4
fi
◇
```

Fragment referenced in 23c.

Create/remove a temporary directory to unpack the received tarball. Put the directory in the home filesystem, because then the contents can easily be moved to its final place in the same filesystem.

⟨ *create a tempdir for downloaddir* 24c ⟩ ≡

```
OLDD=‘pwd‘
cd ~/
tmpdir=‘mktemp -d tarbal.XXXXXXX‘
cd $tmpdir
◇
```

Fragment referenced in 23c.
Defines: `tmpdir` 6f, 8c, 9c, 14a, 16e, 22a, 24d.

⟨ *remove the tempdir for downloaddir* 24d ⟩ ≡

```
cd ..
rm -rf $tmpdir
cd $OLDD
◇
```

Fragment referenced in 23c.
Uses: `tmpdir` 24c.

The tarball ought to contain a directory-tree with as root the name of the submitter, that has been given as argument. This directory ought to be moved to its right place. However, this is a dangerous operation. Every subdirectory could be overwritten if the root directory has a wrong name. Therefore, we have list of allowed names, and compare the name of the imported root with the list.

Note, that the `mv` operation does not move a directory if the target directory already exists. Therefore we have to remove the target directory first. Users must keep a clone of the directory tree and upload modificated clones of the complete directory tree.

So, proceed as follows: 1) unpack the tarball in the temporary directory; 2) Find the name of the root directory; 3) compare with a list of allowed names; 4) remove the target directory if it exists; 5) move the uploaded directory to its proper place; 6) echo “OK” when this was successfull and “error” otherwise.

⟨ *test and unpack the tarball* 25a ⟩ ≡

```
tar -xzf - 2>/dev/null
EXITSTATE=$?
if
  [ $EXITSTATE -eq 0 ]
then
  ⟨ get the name of the root directory 25d ⟩
  ⟨ check whether the rootname is valid 25e ⟩
  if $ISVALIDROOT
  then
    ⟨ replace the target directory by the upload 26a ⟩
  else
    ⟨ write log (25b directory $ROOTNAME not valid ) 7a ⟩
    EXITSTATE=4
  fi
else
  ⟨ write log (25c no valid tarball received ) 7a ⟩
fi
◇
```

Fragment referenced in 23c.

⟨ *get the name of the root directory* 25d ⟩ ≡

```
ROOTNAME=‘ls -1‘
ROOTNAME=${ROOTNAME%%/*}
◇
```

Fragment referenced in 25a.
Defines: ROOTNAME 25be, 26a.

⟨ *check whether the rootname is valid* 25e ⟩ ≡

```
ISVALIDROOT=false
while read user remainder
do
  if
    [ "$user" = "$ROOTNAME" ]
  then
    ISVALIDROOT=true
  fi
done  < /home/phuijgen/nlp/service/userlist
◇
```

Fragment referenced in 25a.
Uses: ROOTNAME 25d.

⟨ *replace the target directory by the upload* 26a ⟩ ≡
```
rm -rf /home/phuijgen/nlp/service/$ROOTNAME
if
  mv $ROOTNAME /home/phuijgen/nlp
 2>/dev/null
then
  ⟨ write log (26b wrote directory $DIRROOT ) 7a ⟩
else
  ⟨ write log (26c directory $DIRROOT not written ) 7a ⟩
  EXITSTATE=4
fi
◇
```
Fragment referenced in 25a.

A Translate and run

This chapter assembles the Makefile for this project.

"Makefile" 26d≡
```
⟨ default target 26e ⟩

⟨ parameters in Makefile 3a, ... ⟩

⟨ impliciete make regels 28b ⟩
⟨ expliciete make regels 3d, ... ⟩
⟨ make targets 28a, ... ⟩
◇
```

The default target of make is `all`.

⟨ *default target* 26e ⟩ ≡
```
all : ⟨ all targets 26f ⟩
.PHONY : all

◇
```
Fragment referenced in 26d.
Defines: `all` Never used, `PHONY` Never used.

One of the targets is certainly the PDF version of this document.

⟨ *all targets* 26f ⟩ ≡
```
alpinist.pdf◇
```
Fragment referenced in 26e.
Uses: `pdf` 28a.

We use many suffixes that were not known by the C-programmers who constructed the `make` utility. Add these suffixes to the list.

⟨ *parameters in Makefile* 27a ⟩ ≡
```
.SUFFIXES: .pdf .w .tex .html .aux .log
```

◇

Fragment defined by 3ac, 27a, 28c, 30c, 31c.
Fragment referenced in 26d.
Defines: `SUFFIXES` Never used.
Uses: `pdf` 28a.

A.1 Pre-processing

To make usable things from the raw input `a_alpinist‘’.w`, do the following:

1. Process `$` characters.
2. Run the m4 pre-processor.
3. Run nuweb.

This results in a LaTeX file, that can be converted into a PDF or a HTML document, and in the program sources and scripts.

A.1.1 Process dollar characters

Many “intelligent” TeX editors (e.g. the auctex utility of Emacs) handle `$` characters as special, to switch into mathematics mode. This is irritating in program texts, that often contain `$` characters as well. Therefore, we make a stub, that translates the two-character sequence `\$` into the single `$` character.

⟨ *expliciete make regels* 27b ⟩ ≡
```
m4_alpinist.w : a_alpinist.w
        gawk '{if(match($$0, "@%")) {printf("%s", substr($$0,1,RSTART-1))} else print}' a_alpinist.w \
          | gawk '{gsub(/[\\][\$$]/, "$$");print}'  > m4_alpinist.w
```

◇

Fragment defined by 3d, 27bc, 28d, 31d.
Fragment referenced in 26d.
Uses: `print` 28a.

Run the M4 pre-processor:

⟨ *expliciete make regels* 27c ⟩ ≡
```
alpinist.w : m4_alpinist.w inst.m4 texinclusions.m4
        m4 -P m4_alpinist.w > alpinist.w
```

◇

Fragment defined by 3d, 27bc, 28d, 31d.
Fragment referenced in 26d.

A.2 Typeset this document

Enable the following:

1. Create a PDF document.
2. Print the typeset document.
3. View the typeset document with a viewer.

In the three items, a typeset PDF document is required or it is the requirement itself.

Make a PDF document.

⟨ *make targets* 28a ⟩ ≡
```
pdf : alpinist.pdf

print : alpinist.pdf
        lpr alpinist.pdf

view : alpinist.pdf
        xpdf alpinist.pdf
```
◇
Fragment defined by 28a, 31ab.
Fragment referenced in 26d.
Defines: `pdf` 26f, 27a, 28b, 29a, `print` 5a, 7c, 8ab, 9bc, 10b, 16ce, 19l, 20acd, 27b, `view` Never used.

Create the PDF document. This may involve multiple runs of nuweb, the LaTeX processor and the bibTeX processor, and dpends on the state of the `aux` file that the LaTeX processor creates as a by-product. Therefore, this is performed in a separate script, `w2pdf`.

A.2.1 The w2pdf script

The three processors nuweb, LaTeX and bibTeX are intertwined. LaTeX and bibTeX create parameters or change the value of parameters, and write them in an auxiliary file. The other processors may need those values to produce the correct output. The LaTeX processor may even need the parameters in a second run. Therefore, consider the creation of the (PDF) document finished when none of the processors causes the auxiliary file to change. This is performed by a shell script `w2pdf`

Note, that in the following `make` construct, the implicit rule `.w.pdf` is not used. It turned out, that make did not calculate the dependencies correctly when I did use this rule.

⟨ *impliciete make regels* 28b ⟩ ≡
```
%.pdf : %.w $(W2PDF)
        chmod 775 $(W2PDF)
        $(W2PDF) $*
```
◇
Fragment referenced in 26d.
Uses: `pdf` 28a.

⟨ *parameters in Makefile* 28c ⟩ ≡
```
W2PDF=./w2pdf
```
◇
Fragment defined by 3ac, 27a, 28c, 30c, 31c.
Fragment referenced in 26d.

⟨ *expliciete make regels* 28d ⟩ ≡
```
$(W2PDF) : alpinist.w
        nuweb alpinist.w
```
◇
Fragment defined by 3d, 27bc, 28d, 31d.
Fragment referenced in 26d.
Uses: `nuweb` 30a.

"w2pdf" 29a≡
```
#!/bin/bash
# w2pdf -- make a pdf file from a nuweb file
# usage: w2pdf [filename]
#  [filename]: Name of the nuweb source file.
# 20130310 at 2206h: Generated by nuweb from a_alpinist.w
echo "translate " $1 >w2pdf.log
⟨ filenames in w2pdf 29c ⟩

⟨ perform the task of w2pdf 29b ⟩

◇
```
Uses: nuweb 30a, pdf 28a.

The script retains a copy of the latest version of the auxiliary file. Then it runs the three processors nuweb, LaTeX and bibTeX, until they do not change the auxiliary file.

⟨ perform the task of w2pdf 29b ⟩ ≡
```
⟨ run the processors until the aux file remains unchanged 30b ⟩
⟨ remove the copy of the aux file 29d ⟩
◇
```
Fragment referenced in 29a.

The user provides the name of the nuweb file as argument. Strip the extension (e.g. `.w`) from the filename and create the names of the LaTeX file (ends with `.tex`), the auxiliary file (ends with `.aux`) and the copy of the auxiliary file (add `old.` as a prefix to the auxiliary filename).

⟨ filenames in w2pdf 29c ⟩ ≡
```
nufil=$1
trunk=${1%%.*}
texfil=${trunk}.tex
auxfil=${trunk}.aux
oldaux=old.${trunk}.aux
◇
```
Fragment referenced in 29a.
Defines: auxfil 30b, nufil 30a, oldaux 29d, 30b, texfil 30a, trunk 30a.

Remove the old copy if it is no longer needed.

⟨ remove the copy of the aux file 29d ⟩ ≡
```
rm $oldaux
◇
```
Fragment referenced in 29b.
Uses: oldaux 29c.

Run the three processors. Do not use the option `-o` (to suppres generation of program sources) for nuweb, because `w2pdf` must be kept up to date as well.

⟨ *run the three processors* 30a ⟩ ≡
```
nuweb $nufil
pdflatex $texfil
bibtex $trunk
◇
```
Fragment referenced in 30b.
Defines: `bibtex` Never used, `nuweb` 28d, 29a, 31ab, `pdflatex` Never used.
Uses: `nufil` 29c, `texfil` 29c, `trunk` 29c.

Repeat to copy the auxiliary file an run the processors until the auxiliary and a copy do both exist and are equal to each other.

⟨ *run the processors until the aux file remains unchanged* 30b ⟩ ≡
```
while
 ! cmp -s $auxfil $oldaux
do
  if [ -e $auxfil ]
  then
   cp $auxfil $oldaux
  fi
  ⟨ run the three processors 30a ⟩
done
◇
```
Fragment referenced in 29b.
Uses: `auxfil` 29c, `oldaux` 29c.

A.3 Install the service

To install the service on Lisa, the following has to be done:

1. Generate directories;
2. Unpack the nuweb source;
3. Install the program-manager as a “cron job”.
4. Make `alpinomanager` executable.

To begin with the third point: Lisa provides the Unix `crontab` mechanism. In the current installation the author of this program has installed a crontab that runs a script `/home/phuijgen/usrlocal/bin/everyminut` every minute. So, we only have to check whether this script invokes the process-manages and, if this is not the case, to add a line to the script.

⟨ *parameters in Makefile* 30c ⟩ ≡
```
CRONDRIVER=/home/phuijgen/usrlocal/bin/everyminute
ALPMAN=/home/phuijgen/nlp/service/bin/alpinomanager
◇
```
Fragment defined by 3ac, 27a, 28c, 30c, 31c.
Fragment referenced in 26d.
Defines: `CRONDRIVER` Never used.

⟨ *make targets* 31a ⟩ ≡
```
install : alpinist.w $(DIRS)
        nuweb -t alpinist.w
        cd /home/phuijgen/nlp/service/bin && chmod 775 alpinomanager
        cd /home/phuijgen/nlp/service/bin && chmod 775 download_archive
        cd /home/phuijgen/nlp/service/bin && chmod 775 download
        cd /home/phuijgen/nlp/service/bin && chmod 775 upload
        cd /home/phuijgen/nlp/service/bin && chmod 775 filstat
        cd /home/phuijgen/nlp/service/bin && chmod 775 downloaddir
```
◇

Fragment defined by 28a, 31ab.
Fragment referenced in 26d.
Uses: `nuweb` 30a.

A.4 create the program sources

Run nuweb, but suppress the creation of the LaTeX documentation. Nuweb creates only sources that do not yet exist or that have been modified. Therefore make does not have to check this.

⟨ *make targets* 31b ⟩ ≡
```
sources : alpinist.w
        nuweb -t alpinist.w
```
◇

Fragment defined by 28a, 31ab.
Fragment referenced in 26d.
Uses: `nuweb` 30a.

Create the directories. Target `DIRS` is a list of the directories

⟨ *parameters in Makefile* 31c ⟩ ≡
```
MKDIR = mkdir -p
```
◇

Fragment defined by 3ac, 27a, 28c, 30c, 31c.
Fragment referenced in 26d.

⟨ *expliciete make regels* 31d ⟩ ≡
```
$(DIRS) :
        $(MKDIR) $(DIRS)
```
◇

Fragment defined by 3d, 27bc, 28d, 31d.
Fragment referenced in 26d.

B Indexes

C Filenames

`"../bin/alpi"` Defined by 12b.

"../bin/alpinomanager" Defined by 18b.
"../bin/download" Defined by 21a.
"../bin/downloaddir" Defined by 23c.
"../bin/download_archive" Defined by 21h.
"../bin/filstat" Defined by 22b.
"../bin/upload" Defined by 23a.
"Makefile" Defined by 26d.
"w2pdf" Defined by 29a.

D Macro's

⟨ all targets 26f ⟩ Referenced in 26e.
⟨ apply the alpino command 16a ⟩ Referenced in 14a.
⟨ awk script to compare the active-jobs list with the jobcounter list 10b ⟩ Referenced in 10a.
⟨ cd naar werkdir 21g ⟩ Referenced in 21ah.
⟨ change "wait" to "run" in jobcountfile 8a ⟩ Referenced in 12b.
⟨ check whether the rootname is valid 25e ⟩ Referenced in 25a.
⟨ check whether update is necessary 6a ⟩ Referenced in 19bg.
⟨ check/perform every time 10c ⟩ Referenced in 19a.
⟨ check/perform frequent tasks 19g ⟩ Referenced in 19a.
⟨ check/perform now-and-then tasks 19b ⟩ Referenced in 19a.
⟨ compare the active-jobs list with the jobcounter list 10a ⟩ Referenced in 9c.
⟨ compare the logfile list with the jobcounter list 9b ⟩ Referenced in 8c.
⟨ construct the alpino command 16c ⟩ Referenced in 14a.
⟨ create a tempdir for downloaddir 24c ⟩ Referenced in 23c.
⟨ create name for tempfile 6f ⟩ Referenced in 12b, 18b, 21ah, 23a.
⟨ decrement filecontent 5c ⟩ Referenced in 13b.
⟨ default target 26e ⟩ Referenced in 26d.
⟨ define variables for filenames 3b, 7b, 13e ⟩ Referenced in 12b, 18b, 21ah, 22b, 23a.
⟨ derive number of jobs to be submitted 11f ⟩ Referenced in 11a.
⟨ determine number of running processes 20c ⟩ Referenced in 20a.
⟨ do brief check of expired jobs 8c ⟩ Referenced in 8f.
⟨ do not run when the intray is empty 11g ⟩ Referenced in 12b.
⟨ do the frequent tasks 8f, 20e ⟩ Referenced in 19g.
⟨ do the now-and-then tasks 9e, 20f ⟩ Referenced in 19b.
⟨ echo exitstate 23g ⟩ Referenced in 23c.
⟨ expliciete make regels 3d, 27bc, 28d, 31d ⟩ Referenced in 26d.
⟨ fetch a file from the intray 15c ⟩ Referenced in 15a.
⟨ filenames in w2pdf 29c ⟩ Referenced in 29a.
⟨ find out the job number 7e ⟩ Referenced in 12b.
⟨ functions in alpars 15ad ⟩ Referenced in 12b.
⟨ generate format for time command 14g ⟩ Referenced in 14a.
⟨ get the argument of the downloaddir script 24a ⟩ Referenced in 23c.
⟨ get the name of the root directory 25d ⟩ Referenced in 25a.
⟨ impliciete make regels 28b ⟩ Referenced in 26d.
⟨ increment filecontent 5b ⟩ Referenced in 13b.
⟨ init logfile 6g ⟩ Referenced in 12b, 18b, 21ah, 22b, 23ac.
⟨ load sara modules 12f ⟩ Referenced in 12b.
⟨ log time lapse 14f ⟩ Referenced in 14a.
⟨ look for an xml-treebank request 16e ⟩ Referenced in 16c.
⟨ make a list of jobs that produced logfiles 9a ⟩ Referenced in 8c.
⟨ make list of tray 17o ⟩ Referenced in 17e.
⟨ make targets 28a, 31ab ⟩ Referenced in 26d.
⟨ pack/send the xml-directory and remove xml directory 17d ⟩ Referenced in 17a.
⟨ parameters in Makefile 3ac, 27a, 28c, 30c, 31c ⟩ Referenced in 26d.
⟨ PBS job parameters 12a ⟩ Referenced in 12b.
⟨ perform a single process 14a ⟩ Referenced in 13b.

⟨ perform the management tasks 19a ⟩ Referenced in 18b.
⟨ perform the task of w2pdf 29b ⟩ Referenced in 29a.
⟨ print notification 17e ⟩ Referenced in 22b.
⟨ remove old files from the outtray 20d ⟩ Referenced in 20f.
⟨ remove the copy of the aux file 29d ⟩ Referenced in 29b.
⟨ remove the job from the counter 8b ⟩ Referenced in 12b.
⟨ remove the tempdir for downloaddir 24d ⟩ Referenced in 23c.
⟨ replace files from proctray when no processes are running 20a ⟩ Referenced in 10c.
⟨ replace the target directory by the upload 26a ⟩ Referenced in 25a.
⟨ report processors 18a ⟩ Referenced in 17e.
⟨ restore old files from the process-tree 19l ⟩ Referenced in 20e.
⟨ run the processors until the aux file remains unchanged 30b ⟩ Referenced in 29b.
⟨ run the three processors 30a ⟩ Referenced in 30b.
⟨ send the output to the sender 17a ⟩ Referenced in 14a.
⟨ set environments for the NLP applications 12g, 13a ⟩ Referenced in 12b.
⟨ start parallel processes that perform the parsing 13b ⟩ Referenced in 12b.
⟨ start the alpinomanager 18f ⟩ Referenced in 21ah, 22b.
⟨ stop if another alpinomanager is running 18g ⟩ Referenced in 18b.
⟨ submit a job 7c ⟩ Referenced in 11e.
⟨ submit extra jobs 11e ⟩ Referenced in 11a.
⟨ submit jobs when necessary 11a ⟩ Referenced in 10c.
⟨ synchronisation functions 4ab, 5a ⟩ Referenced in 12b, 18b, 21ah, 22b.
⟨ test and unpack the tarball 25a ⟩ Referenced in 23c.
⟨ unpack tar 22a ⟩ Referenced in 21h.
⟨ verify jobs-bookkeeping 9c ⟩ Referenced in 9e.
⟨ write log 7a ⟩ Referenced in 6a, 7c, 11ag, 12b, 16a, 17ae, 18b, 19bg, 21ah, 22b, 23c, 24a, 25a, 26a.

E Variables

`all`: 26e.
`auxfil`: 29c, 30b.
`bibtex`: 30a.
`BOOKKEEPFILE`: 13e, 14f.
`COUNT`: 18g.
`CRONDRIVER`: 30c.
`disparm`: 12f.
`getfile`: 14a, 15d.
`INBAK`: 3b, 11ag, 15cd, 17g, 19l, 20a, 21a, 22a.
`JOBCOUNTFILE`: 7b, 7c, 8ab, 9b, 10a, 11a, 20c.
`LOCKDIR`: 4b, 5a.
`LOGFIL`: 6g, 7a.
`LOGGING`: 6g, 7a.
`nufil`: 29c, 30a.
`nuweb`: 28d, 29a, 30a, 31ab.
`oldaux`: 29c, 29d, 30b.
`passeer`: 4b, 5bc, 6a, 7c, 8ab, 9b, 10a.
`pdf`: 26f, 27a, 28a, 28b, 29a.
`pdflatex`: 30a.
`PHONY`: 26e.
`print`: 5a, 7c, 8ab, 9bc, 10b, 16ce, 19l, 20acd, 27b, 28a.
`PROCBAK`: 3b, 14a, 15d, 17ai, 19l, 20a.
`ROOTNAME`: 25b, 25d, 25e, 26a.
`SUFFIXES`: 27a.
`texfil`: 29c, 30a.
`tmpdir`: 6f, 8c, 9c, 14a, 16e, 22a, 24c, 24d.
`tmpfil`: 6f, 8ab, 9ab, 10a, 21a.
`trunk`: 29c, 30a.

UITBAK: 3b, 17acdk, 20d, 23a.
veilig: 4b, 5bc, 6a, 7c, 8ab, 9b, 10a.
view: 28a.