VLEEM 2

MID-TERM ASSESSMENT REPORT

Annex 6

NUCLEAR CYCLE MODELLING IN VLEEM

Gerhard Kolb, Dag Martinsen
Forschungszentrum Juelich, Systemforschung und Technologische Entwicklung (FZJ/STE)
October 2003

PART 1: General specifications

In order to facilitate the computation of nuclear mass flows, storage contents and waste amounts of nuclear expansion strategies in VLEEM 2 a PC-based Nuclear Mass Balance Model (NMBM) has been developed and tested. The code of NMBM has been written in Microsoft Visual C++ 6.0, but the NMBM runs are executed via exe-files (and some Excel-files) which do not need the implementation of C++ at the user´s PC. The user would need C++ only if he/she would like to read the source codes or to change them.

The model consists logically of 3 parts:

- The capacity programme
- The mass balance programme
- A programme to perform computations with deliberately small time steps based on the results of (more or less) coarse time steps of a size of (normally) some years, e.g., 5 or 10 years within a time horizon of some decades, say a 100 years.
  This 3rd module is mainly of interest,
  1. if the model is part of a comprehensive (superior) energy model with nuclear contributions, and
  2. if this energy model simulates the operation of a plant park in small time steps (months, weeks, days or less).

The NMBM comprises therefore 3 time loops:

1. In the first loop the "coarse" time sequence of individual reactor capacities are calculated via the assumptions of individual capacity additions.
2. In the second loop the corresponding mass balances are calculated, considering operating capacities, capacity additions and retirements.
3. In the third loop the mass balances are calculated with deliberately small (constant) time steps and deliberately chosen load factors for each reactor type and time step (independent from the load factors in the "coarse" calculation where these factors are only reactor-dependent, not variable with time), but based on the dynamic reactor pool as defined for the "coarse" time steps (first and second loops).

The following information material on the NMBM has been prepared which briefly is presented below:

1. NMBM.doc: The Nuclear Mass Balance Model

Describes the structure of the model, the generic input parameters, the definitions of nuclear capacities, of reactor types and their characteristics, the calculation of individual capacities, and the computation of nuclear mass flows based on a fuel cycle divided into five steps:

- Fresh heavy metals: Natural Uranium – U_{nat}, and Thorium – Th
- Fuel element-fabrication - fresh and recycled ones ("re-fabrication")
- Reactor operation
- Interim storage for spent fuel elements, then possibly reprocessing and re-fabrication
- Final repository

The numerical description for each time step goes via 8 "pots" and the final disposal:
Fresh fuel requirements (per time step and cumulated):

- U_{nat}
- Th

Interim storages:

- Fission products (FP)

- Depleted U
- Irradiated U
- Minor actinides (MA)
- Fresh (once recycled) Pu
- Multiple recycled Pu

Final disposal:

- Irradiated (spent) Th and U destined for repository
- Waste: Pu and MA (entire discharges or only their re-fabrication losses), FP, U, Th

For the interim storages initial values ("RESIDs") must be set.
In the case of recycling strategies the required re-fabrication capacities are computed too.

NMBM.doc is included as attachment in the forthcoming VLEEM 2 Monograph on Nuclear Fission (at the latest finished in November).

2. User-Guide.doc: Operational Users´ Guide

Explains how to use the NMBM, lists the requirements for the execution of the NMBM and describes how to start the execution of the model. Four input Excel-files are needed:

- EXCEL-file "vleem2-input" of the basic scenario input data
- EXCEL-file "Matrix.xls" with the specific mass flow data for the reactors
- EXCEL-file "Storage.xls" with initial values for 7 storages and 3 loss factors for re-processing
- EXCEL-file "capf.xls" with capacity factors for computations with "small" time steps

Additionally required are the executable programmes

- "vleem2_mass.exe" (for the nuclear mass flow computations with the "coarse" time step), and
- "vleem2_use.exe" (for the nuclear mass flow computations with the "small" time step, but based on the nuclear reactor structure of the coarse time steps).

These two programmes are the "source code" of the nuclear mass balance model NMBM.

An exe-file "create_charts.exe" is automatically called from *vleem2_mass.exe* and creates Excel-diagrams from the result tables so that the user gets graphical pictures (3 diagrams of capacities and 4 diagrams of nuclear mass flows) of the results with the "coarse" time steps:

Reactor capacities:

1. Capacities divided according to three reactor classes:
   Existing reactors, new reactors for electricity and process heat.
2. Capacities of all reactor types
3. Capacity additions (new capacities) of all reactor types

Nuclear masses and mass flows:

1. Temporal storage contents of discharged Thorium, fission products, minor actinides, fresh and multiple recycled Plutonium
2. Consumption of natural Uranium
3. Consumption of fresh Thorium
4. Refabrication requirements

The Users´Guide identifies also an important consistency check:
The rationale of the main direction in long-term reactor development is the destruction of stockpiles of

- Plutonium (Pu),
- minor actinides (MA),

- some long-lived fission products (FP).

As mentioned above the Pu-stockpile is divided into two segments:
Fresh (once recycled) Pu (Pu-f) and multiple recycled Pu (Pu-rec).
The main consistency checks are twofold:

1. The Pu-stockpiles must not be negative.
2. The total Pu-stockpile (Pu-rec) must not be smaller than the Pu-f stockpile.

Input data prepared for demonstrative purposes, although being more or less realistic for a "High Nuclear Case" for western Europe produce "reasonable" results with regard to the two consistency checks quoted above, as shown below in Figure 1. Figure 2 shows the corresponding capacity distributions. The lack of breeder reactors leads in this demonstration case also to a high consumption of natural U, as Figure 3 demonstrates. Time horizon is 2000-2100, with the "coarse" time steps of 10 years, i.e. 10 periods.

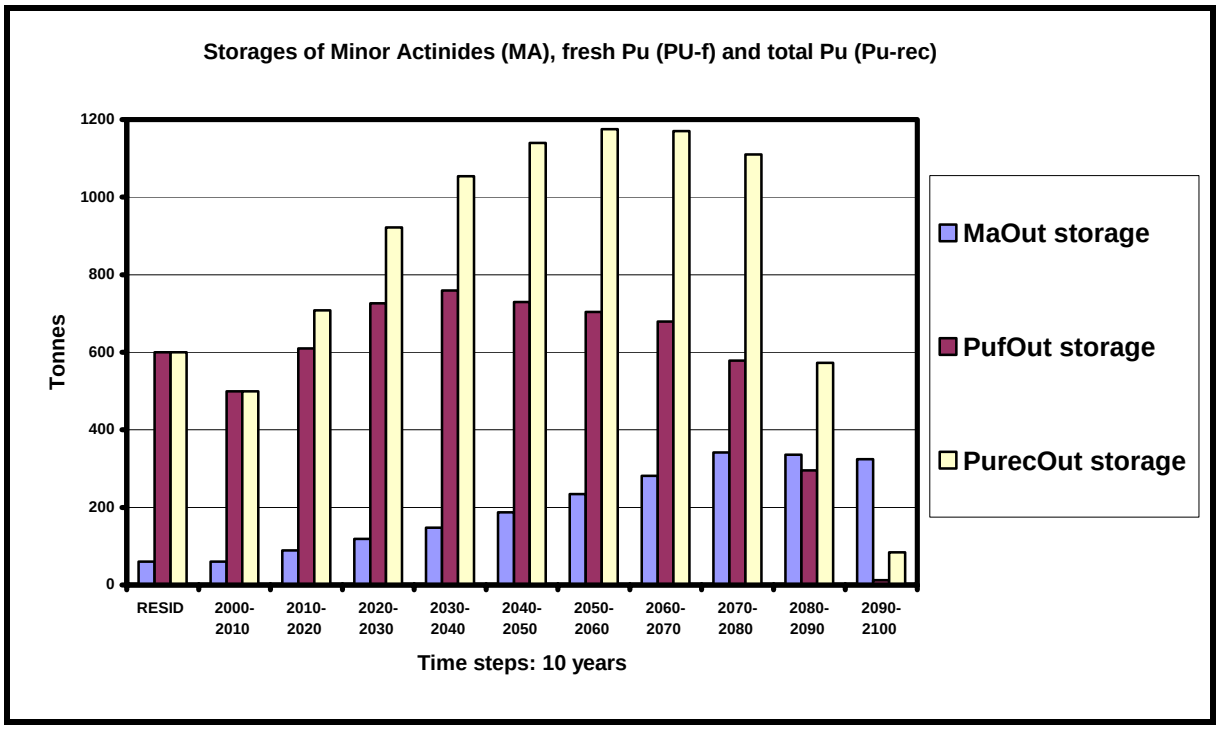


Figure 1: Dynamics of the storages for minor actinides – MA (blue), fresh (and once recycled) Pu – Puf (red), and total Pu – Purec (yellow).

User-Guide.doc contains NMBM.doc as attachment.

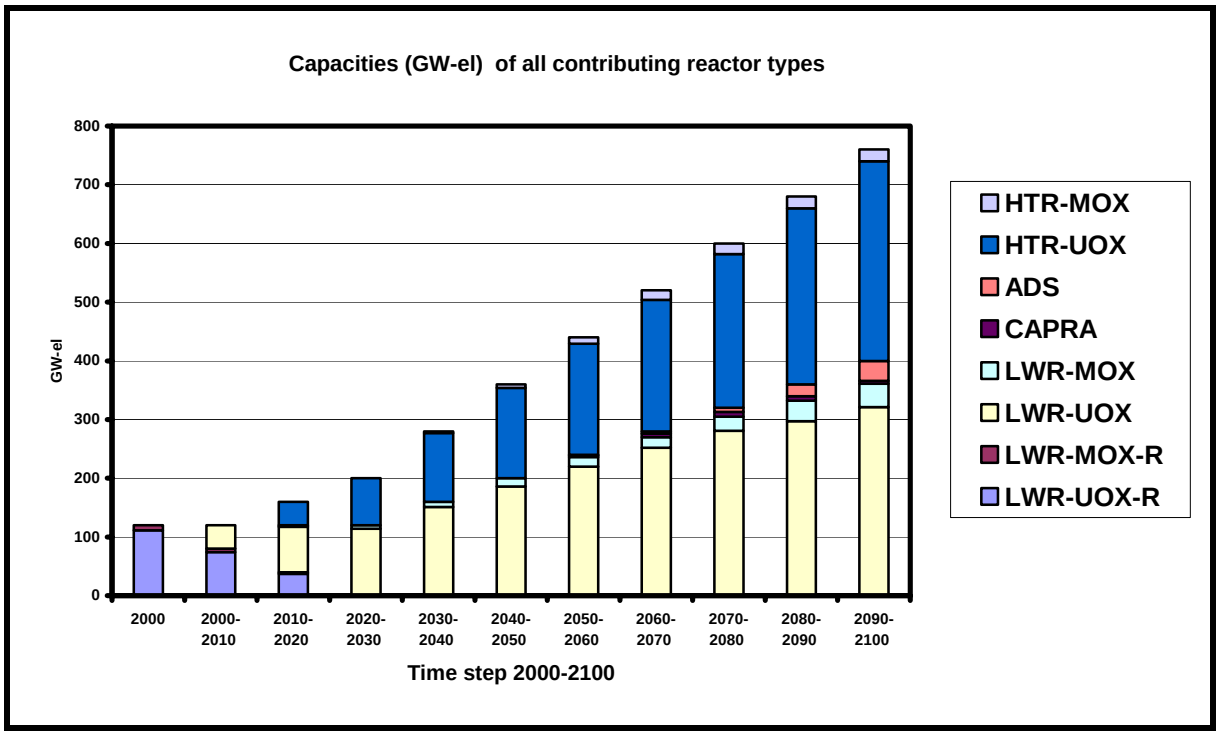


Figure 2: Dynamics of the capacities (GW-el) of 9 contributing reactor types. Observe that the two types marked by "-R" can only die out (as "RESIDs", i.e. the initially already existing reactors), but not be added again.

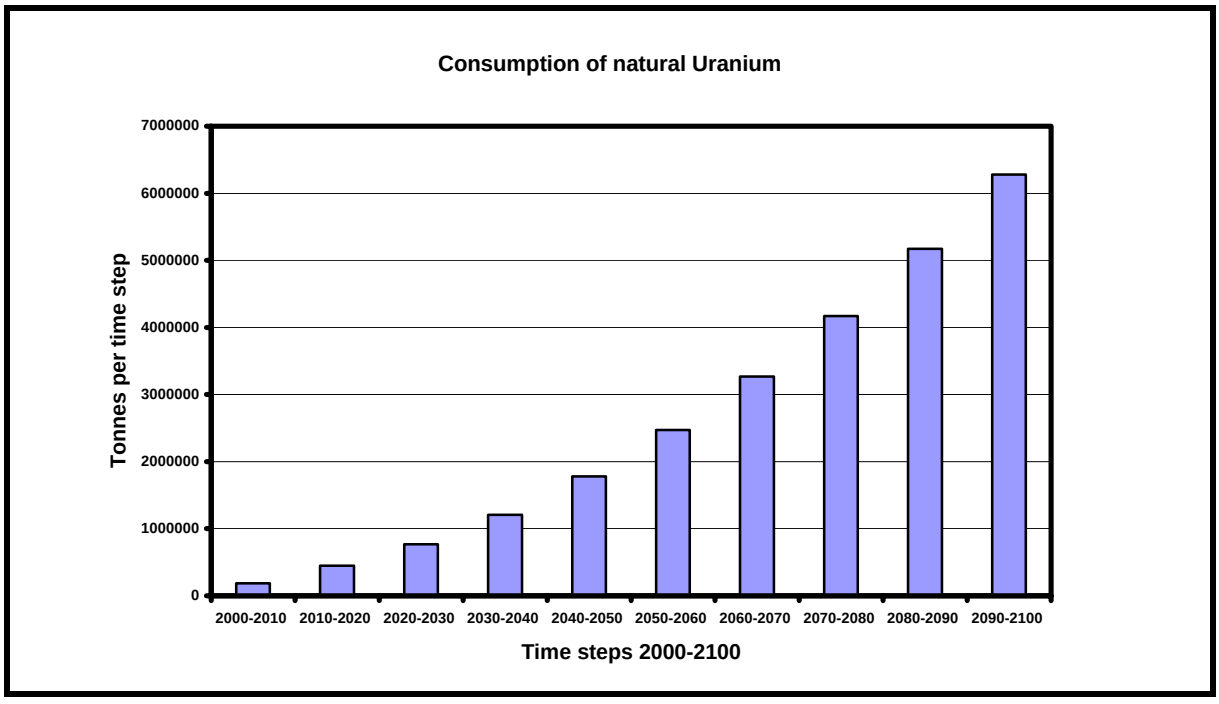


Figure 3: Dynamics of the natural-U consumption of the demonstration strategy. This case apparently leads to a very high U-consumption, nearly twice that much as there are known mineral U-reserves with production costs below 130 US$/kg U (~ 4 million tonnes).

3. Documentation.doc: Documentation of the Nuclear Mass Balance Model

After a brief introduction into its main structure the Documentation describes the two main algorithms (Vleem2_mass.exe and Vleem2_use.exe) and their sub-modules ("components"). These connections of the sub-modules within their main algorithms are also demonstrated graphically in two diagrams. Additionally the Documentation informs about the application of two further auxiliary exe-modules which are executed automatically during the run of the NMBM, but can be used also as stand-alone-programmes:
xls-to_txt.exe and create_charts.exe.
Behind this description the entire C++-source code is enclosed.

User-Guide.doc is attached to Documentation.doc, because it is facilitates for the purpose of better understanding to have a look into the User-Guide during reading the first part of the Documentation.

Part 2: Documentation for the PC-based Nuclear Mass Flow Programmes

VLEEM2_MASS
(Capacity- and Mass Balance-Computations)

and

VLEEM2_USE
(Operational Mass Balance Computations)

Sylvia Gasper, Gerhard Kolb

November 2003

1. Introduction

The software package contains two executable programs **vleem2_mass.exe** and **vleem2_use.exe**, where the second program depends on the results of vleem2_mass.exe.
The following documentation describes the structure and the mode of operation of both parts.
A detailed description of the practical execution and the program structure can be found in the Operational Users´ Guide (referred to below as /OUG/ and containing the Nuclear Mass Balance Model – NMBM – as Annex). It is included here as Attachment.

2. Program description

2.1 VLEEM2_MASS :

The program **vleem2_mass.exe** contains two parts of the application: On the one hand the entire capacity calculation and evaluation of the electricity- and process heat production (the latter expressed as corresponding electricity output via an assumed efficiency) and on the other hand the nuclear mass balance computation.
As input the program requests the time horizon, the various reactor types under consideration per reactor class including their characteristics (here: lifetime, full-load hours per year, out-of-pile times of discharged fuel), the total capacity per class and the individual capacity additions per type in each period. Furthermore a matrix with the specific mass flow data for the reactors is needed for the computation of the mass balance. All the data are checked in form and content.
The program calculates for each time step (period) the individual reactor capacities and the production for each type of reactor as well as incoming and outgoing nuclear materials for each type of reactor, the material storages and the waste. For details see the Attachment.
All the data are displayed in output files, one for the capacity outputs and one for the mass balances, as Excel-tables and (standard) Excel-diagrams.

2.2 VLEEM2_USE :

The program **vleem2_use.exe** performs the same computations as the mass balance program, but for small time steps. Whereas the first program works with rough ("coarse") time steps and calculates the mass flow data for relatively long periods (e.g. 10 years), vleem2_use.exe uses these results and evaluates the mass balances for small fractions of one long time step.
Therefore the program needs capacity factors for each reactor type per small time step in a matrix (capf.xls and capf.txt, respectively).
The results are displayed in an output file as Excel-tables.

2.3 DESCRIPTION OF THE ALGORITHMS :

2.3.1 Vleem2_mass :

The capacity part consists of input, calculation of the individual capacities and production and the output via ***output_cap.xls***.
First the needed information for the time horizon, the reactor types, the total capacity and the initial values are imported into the sub-function *input_cap* and checked on formal correctness.
New_capacity reads in the individual capacity additions and controls if they are conform with the total capacities. In this case it calculates the individual capacities and the (electricity) production per interval for each type of reactor.
The results are written into the output file.
The mass balance part also starts with an input function *input_mass* that reads in the specific nuclear mass flow data matrix and the initial values for storages and waste.
On the basis of these inputs and the results of the capacity part the mass balances, the storages and the waste are calculated for each rough time step in the sub-functions *sub_in*, *sub_out* and *waste_calc*.
In *output_mass* these results are displayed in the file ***output_mass.xls*** as Excel-tables and Excel-diagrams.
At the end of the program *error_output* checks, if an error has occurred during the execution and if this is the case then it creates an error-output file ***error_output.txt***.
If the program ran without an error, the data that are needed for the second program are recorded in another output file *mass_results.txt*. This file is only used for the input of *vleem2_use.*

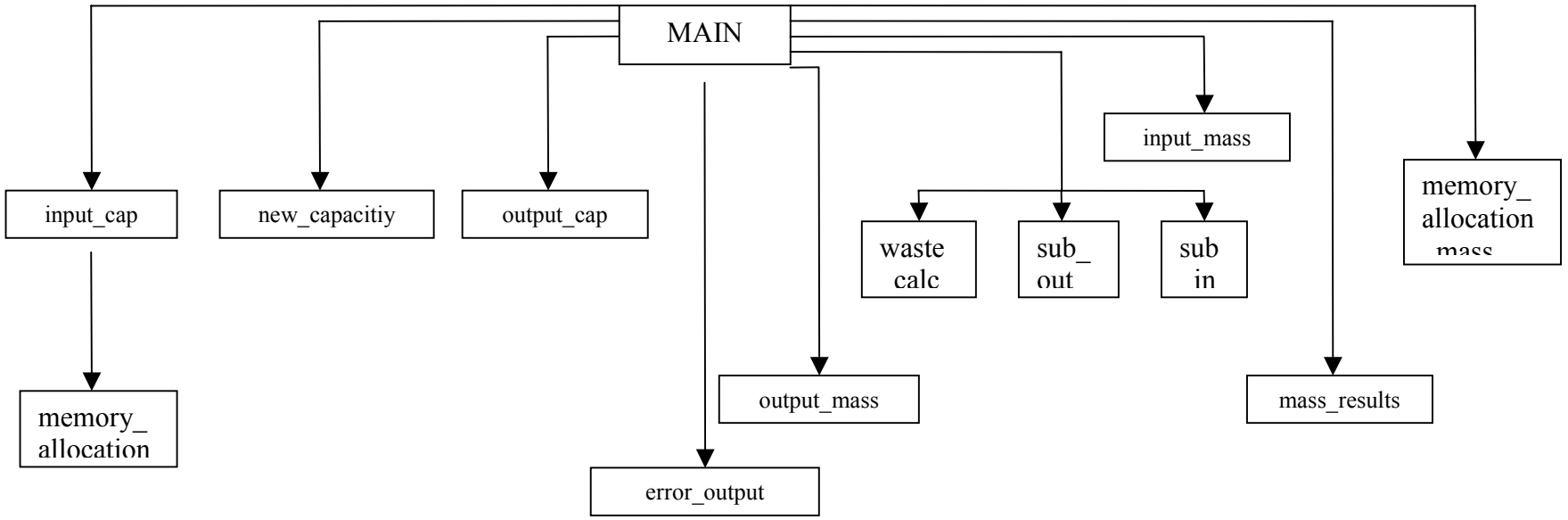
2.3.2 Vleem2_use :

Vleem2_use.exe is structured in nearly the same way as the mass balance part of the first program. It starts with the import of *mass_results.txt* to get the mass flow data and the initial values of the calculation for rough time steps. Then the user has to specify the number of short time steps per interval; the needed capacity factors for the short periods are read in from ***capf.xls***.
If the input was formally correct, the computations for the nuclear mass flows are made in reference to the short time steps as in vleem2_mass.exe for long periods. The calculations proceed in the sub-functions *mass_calculation* and *waste_calculation.*
The results for the short time intervals are written in the output file ***output_use.xls***.
Also vleem2_use.exe checks every input and calculation and aborts execution if an error occurs. In this case ***error_output.txt*** is created that describes the this error.

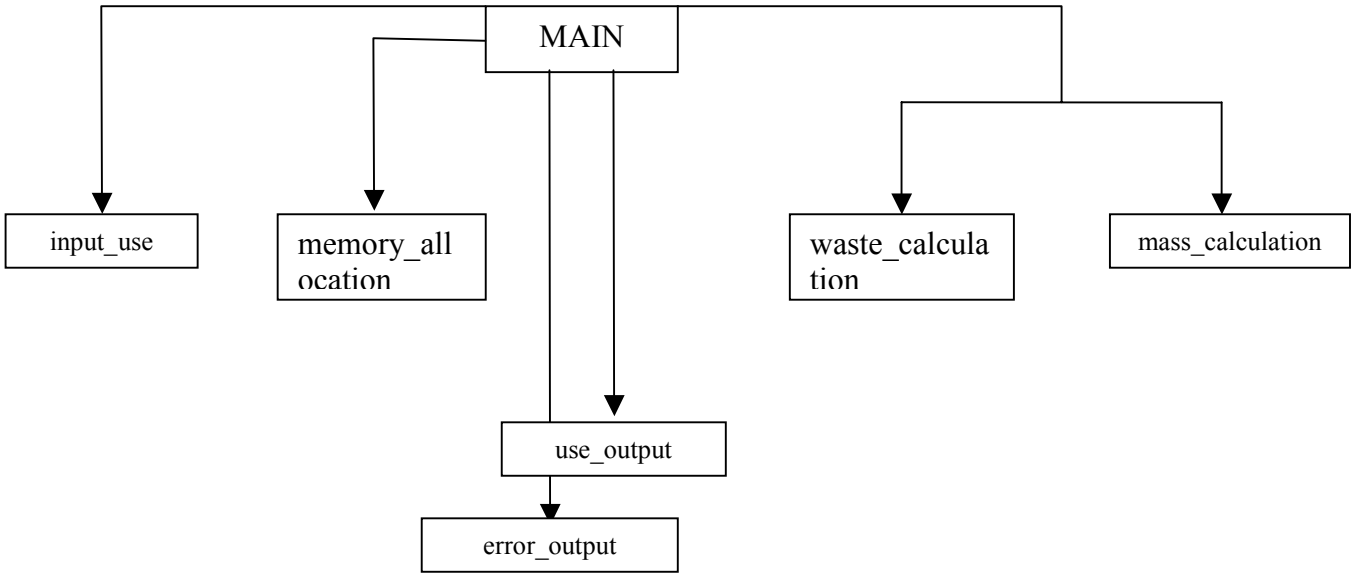
The two subsequent diagrams show the information flows of these two algorithms (inputs and results).

2.3 PROGRAM STRUCTURE

2.3.1 Vleem2_mass :



2.3.2 Vleem2_use :



3. Description of the components

3.1 VLEEM2_MASS :

3.1.1 main
The main function controls the application flow of both program parts. From there the in- and output functions, the calculation functions and the error output are called.
After every sub-function the program is checked on errors and aborted, if anything incorrect has occurred.

3.1.2 input_cap
Input_cap is the input function for the needed data at the beginning of the program execution. It reads in the time horizon, the characteristics of the different reactor types (here: lifetime, full-load hours per year, out-of-pile times of discharged fuel), the provided total capacities for each reactor class and the initial values for each type.

3.1.3 memory_allocation
This function serves as allocation of memory for all used variables in the capacity part and depends on the user´s input in the previous function.

3.1.4 new_capacity
New_capacity reads in the individual capacity additions respectively the decommissioning (shut-downs) and calculates the single capacities for each type of reactor in each period as well as their (electricity) production.

3.1.5 output_cap
The output function creates an output file and prints the results concerning individual capacities and production received in new_capacity.

3.1.6 input_mass
In input_mass the content of the file containing the mass flow data matrix is saved internally. Furthermore it reads in the initial values for the storage and the loss factors of reprocessing from a second input file.

3.1.7 memory_allocation
To allocate the storage for the previous described data memory_allocation is called from the main function.

3.1.8 sub_in, sub_out, waste_calc
These three functions serve as processing of the input data and calculate the material cycles, so the loaded and discharged nuclear materials and the amounts of wastes.

3.1.9 mass_results, output_mass
Both are output functions. mass_results is only created for the input of the second program vleem2_use and contains special values which are needed there.
Output_mass is the proper output file of the mass balance part created for the user including all the interesting data of the previous calculation in sub_in, sub_out and waste_calc.

3.1.10 error_output
This function only gets importance, if an error has occurred during the execution. In this case it creates an error file and prints out the corresponding problem report.

3.2 VLEEM2_USE

3.2.1 main
The main function controls the application flow of both program parts. From there the in- and output functions, the calculation functions and the error output are called.
After every sub-function the program is checked on errors and aborted, if anything incorrect has occurred.

3.2.2 input_use
First the input function gets the results of the first program that are needed for the evaluation of the mass balances for short time steps out of the result file created by mass_results.
The second input file that is handled by input_use contains a matrix with capacity factors for each reactor type and interval.

3.2.3 waste_calculation, mass_calculation
These functions have similar jobs as sub_in, sub_out and waste_calc in vleem2_mass, but calculate the data for short intervals.

3.2.4 use_output
Use_output creates an output file analogous to the mass balance output with all the calculated mass flows and wastes for each reactor type.

3.2.5 error_output
This function only gets importance, if an error has occurred during the execution. In this case it creates an error file and prints out the corresponding problem report.

4. Information about the other components of the software package

4.1 XLS_TO_TXT.EXE :

This program saves excel-files (.xls) as text- (.txt) or csv- (.csv) files and has to be executed every time the input files have been changed.
The user can choose the file or files he wants to save from a dialog box.

4.2 CREATE_CHARTS.EXE :

Create_charts.exe is automatically called from *vleem2_mass.exe* and creates Excel-diagrams for output_cap and output_mass from the result tables so that the user gets graphical pictures (3 diagrams of capacities and 4 diagrams of nuclear mass flows) of the results with the "coarse" time steps:
Reactor capacities:

4. Capacities divided according to the (three) reactor classes
5. Capacities of all reactor types
6. Capacity additions (new capacities) of all reactor types

Nuclear masses and mass flows:

5. Temporal storage contents of minor actinides, fresh and multiple recycled Plutonium
6. Consumption of natural Uranium
7. Consumption of fresh Thorium
8. Refabrication requirements

4.3 EXECUTE.BAT :

The batch-file execute.bat contains the executions of the programs vleem2_mass.exe and vleem2_use.exe. It simplifies the execution with all the parameters. In its code the names of the input files have to be changed if other names for the input files as the standard ones are used. In this case its code has to be saved and the file closed before the next execution of execute.bat.

More detailed information about these components and their handling you can find in the O*perational Users´ Guide /OUG/* (included as Attachment).

5. Source Code

5.1 VLEEM2_MASS :

/* filename: vleem2_header.h */

```
#ifndef VLEEM2_HEADER
#define VLEEM2_HEADER

#include "vleem2_capacity_header.h"
#include "vleem2_mass_header.h"

#define INPUT_ANSWER_ERROR 21

/* float to string */
void dezkomma(char* str, float wert);


/* function for global error output */
void error_output(void);

/* results of part I and II for input of part III */
void mass_results(int ttypes, reactor_class** classes, float**** TMatin1, float**** TMatin2,
float**** TMatout1, float**** TMatout2, float** StorageResid,
                                  float UirrWasteResid, float PuWasteResid, float MaWasteResid,
float VPU, float VUIRR, float VMA);


#endif
```

/* filename: vleem2_capacity_header.h */

```
#ifndef VLEEM2_CAPACITY_HEADER
#define VLEEM2_CAPACITY_HEADER

/* include-files */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <ctype.h>

/*  enumeration of the different classes of reactors  */
typedef enum{RESID=0, STR, PW} class_number;

/*  enumeration of the different types in a class of reactors  */
typedef int type_number;

/*  new type for a reactor-type  */
typedef struct{
       char* name;
       type_number nr;
       int lifetime;
       float full_load_hour;
       int cooling_time;
} reactor_type;

/*  new type for a reactor-class  */
typedef struct{
       char* name;
       class_number nr;
       int number_of_types;
       reactor_type* types;
} reactor_class;


/*  time horizon  */
int JI;             /* year of beginning */
int JF;             /* year of ending */
int JT;             /* length of one time interval */
int IT;             /* number of time intervals ((JF-JI)/JT) */

/*  variables for error  */
int error;
int error_int;
int error_class;

/*  variable for difference between planned and reached capacity in one interval */
float difference;


/*  errors  */
#define ARGUMENT_ERROR 1
#define OPEN_ERROR 2
#define INPUT_TH_ERROR 3
#define INPUT_TYPENO_ERROR 4
#define INPUT_FORMAT_ERROR 5
#define INPUT_NAME_ERROR 6
#define INPUT_LT_ERROR 7
#define INPUT_FLH_ERROR 8
#define INPUT_CT_ERROR 9
#define INPUT_LT_FLH_ERROR 10
#define INPUT_TTC_ERROR 11
#define INPUT_TCC_ERROR 12
#define INPUT_NEWCAP_ERROR 13
#define INPUT_NEG_RESID_ERROR 14
#define INPUT_POSDIFF_ERROR 15
#define INPUT_NEGDIFF_ERROR 16




/*  input-function  */
void input_cap(FILE* infile, reactor_class** classes, float*** TotalClassCap,
                  float**** NewTypeCap, float**** TotalTypeCap, float**** Production);

/*  function for new capacities  */
void new_capacity(FILE* infile, reactor_class** classes, float*** TotalClassCap,
```

```
                float**** NewTypeCap, float**** TotalTypeCap, float**** Production);

/*  function for memory allocation  */
void memory_allocation_cap(reactor_class** classes, float*** TotalClassCap,
                float**** NewTypeCap, float**** TotalTypeCap, float**** Production);


/*  functions for calculation  */

/* total capacity for class 'r_class' after period 'interval' */
float tcap(reactor_class* r_class, int interval, float*** TotalClassCap);

/* new capacity for type 'r_type' of class 'r_class' in period 'interval' */
float cap_new(reactor_class* r_class, reactor_type* r_type, int interval, float****
NewTypeCap);

/* shut down for type 'r_type' of class 'r_class' in period 'interval' */
float still(reactor_class* r_class, reactor_type* r_type, int interval, float**** NewTypeCap);

/* total capacity for class 'r_class' after period 'interval' with topical input */
float tcap_test(reactor_class* r_class, int interval, float*** TotalClassCap, float****
NewTypeCap);

/* difference between tcap and tcap_test */
float diff(reactor_class* r_class, int interval, float*** TotalClassCap, float****
NewTypeCap);

/* total capacity for type 'r_type' of class 'r_class' after period 'interval' */
float cap(reactor_class* r_class, reactor_type* r_type, int interval, float**** NewTypeCap,
float**** TotalTypeCap);

/* production for type 'r_type' of class 'r_class' after period 'interval' */
float pro(reactor_class* r_class, reactor_type* r_type, int interval, float**** NewTypeCap,
float**** TotalTypeCap);

/*  function for output  */
void output_cap(FILE* outfile, reactor_class** classes, float*** TotalClassCap,
                float**** NewTypeCap, float**** TotalTypeCap, float**** Production);

#endif
```

/* filename: vleem2_mass_header.h */

```
#ifndef VLEEM2_MASS_HEADER

#define VLEEM2_MASS_HEADER

/*  enumeration of the substances  */
typedef enum{UnatIn = 1, ThIn, FpIn, UdeplIn, UirrIn,
                      MaIn, PufIn, PurecIn, HmIn, ThUOut,
                      FpOut, UdeplOut, UirrOut, MaOut, PufOut,
                      PurecOut} substance;

/*  errors  */
#define INPUT_MATRIX_ERROR 17
#define INPUT_STORAGE_ERROR 18
#define INPUT_FACTOR_ERROR 19
#define INPUT_WASTE_ERROR 20

/*  function for memory allocation  */
void memory_allocation_mass(int ttypes, float*** Matrix,float*** Matin, float**** TMatin1,
float**** TMatin2,
                                                    float**** TMatout1,   float**** TMatout2
,float**** TMatout,
                                                    float*** Storage, float** Refab, float**
Unat, float** Unatt, float** Tht,
                                                    float** UirrWaste, float** PuWaste,
float** MaWaste, float** Waste);

/*  input function  */
void input_mass(FILE* infile_mass1, FILE* infile_mass2, int ttypes, float*** Matrix,
                            float*** Storage, float** StorageResid, float** UirrWaste,
float** PuWaste, float** MaWaste,
                           float* UirrWasteResid, float* PuWasteResid, float* MaWasteResid,
                           float* VPU, float* VUIRR, float* VMA);

/* calculation of incoming substances */
void sub_in(int ttypes, float*** Matrix, float*** Matin, float**** TMatin1, float**** TMatin2,
float** Refab,
                      float** Unat, float** Unatt, float** Tht, float**** Production,
                      float**** NewTypeCap, reactor_class** classes);

/* calculation of outcoming substances */
void sub_out(int ttypes, float*** Matrix, float*** Matin, float**** TMatout1,
                       float**** TMatout2 ,float**** TMatout, float**** Production,
                       float**** NewTypeCap, float*** Storage, reactor_class** classes);

/* calculation of waste */
void waste_calc(int ttypes, float*** Matin, float*** Storage, float** UirrWaste, float**
PuWaste,
                            float** MaWaste, float** Waste, float VPU, float VUIRR, float
VMA);

/* output */
void output_mass(FILE* outfile, reactor_class** classes, int ttypes, float*** Matin, float****
TMatin1, float**** TMatin2, float**** TMatout, float*** Storage,
                              float** Refab, float** Unat, float** Unatt, float** Tht,
float** StorageResid, float** UirrWaste, float UirrWasteResid, float** PuWaste,
                              float PuWasteResid, float** MaWaste, float MaWasteResid,
float** Waste);

#endif
```

/* filename: vleem2_capacity.c */

```
#include "vleem2_header.h"
#include "vleem2_capacity_header.h"
#include "vleem2_mass_header.h"


/*  time horizon  */
int JI;             /* year of beginning */
int JF;             /* year of ending */
int JT;             /* length of one period of time */
int IT;             /* number of periods ((JF-JI)/JT) */


/*  variable for error  */
int error;
int error_int;
int error_class;

/*  variable for difference between planned and reached capacity in one interval */
float difference;


int main(int argc, char** argv)
{

        /*  part I  */
        /*  three reactor-classes for RESID, STR, PW  */
        reactor_class* classes;

        /*  total capacity per interval for each reactor-class  */
        float **TotalClassCap;

        /*  new capacity / closure per interval for each reactor-type of each reactor-class  */
        float ***NewTypeCap;

        /*  total capacity per interval for each reactor-type of each reactor-class  */
        float ***TotalTypeCap;

        /*  production per interval for each type of classes STR and PW  */
        float ***Production;

        /*  part II  */
        /*  total number of types  */
        int ttypes;

        /*  matrix for part II  */
        float** Matrix;

        /*  total incoming substance for each substance per period  */
        float** Matin;

        /*  incoming substance for each type and each material per period  */
        float*** TMatin1;
        float*** TMatin2;

        /*  refabrication per period  */
        float* Refab;

        /*  outcoming substance for each type and each material per period  */
        float*** TMatout1;
        float*** TMatout2;
        float*** TMatout;

        /*  storage for each substance per period  */
        float* StorageResid;
        float** Storage;

        /*  consume of natural uran per period  */
        float* Unat;

        /*  consume of cumulated natural uran per period  */
        float* Unatt;

        /*  consume of cumulated thorium per period  */
        float* Tht;

        /*  Uirr waste  */
```

```
	float* UirrWaste;
	float UirrWasteResid;

	/*  Pu waste  */
	float* PuWaste;
	float PuWasteResid;

	/*  Ma waste  */
	float* MaWaste;
	float MaWasteResid;

	/*  total waste per period  */
	float* Waste;

	/*  loss factors  */
	float VPU, VUIRR, VMA;

	FILE* infile_cap;
	FILE* infile_mass1;
	FILE* infile_mass2;
	FILE* outfile_cap;
	FILE* outfile_mass;


	if(argc!=4)
		error = ARGUMENT_ERROR;

	else
	{
		/*  file input  */
		if((infile_cap = fopen(argv[1], "r"))==NULL)
			error = OPEN_ERROR;

		if((infile_mass1 = fopen(argv[2], "r"))==NULL)
			error = OPEN_ERROR;

		if((infile_mass2 = fopen(argv[3], "r"))==NULL)
			error = OPEN_ERROR;

	}


	if(error == 0)
	{
		if((outfile_cap = fopen("output_cap.xls", "w")) ==NULL)
			error = OPEN_ERROR;

		if((outfile_mass = fopen("output_mass.xls", "w")) ==NULL)
			error = OPEN_ERROR;


		/* capacity part */
		if(error == 0)
		{

			assert(classes = (reactor_class*) malloc(3*sizeof(reactor_class)));
			/*  set name of reactor-classes  */
			classes[0].name = "RESID";
			classes[1].name = "STR";
			classes[2].name = "PW";

			/*  set class_number of reactor-classes  */
			classes[0].nr = RESID;
			classes[1].nr = STR;
			classes[2].nr = PW;

			/* input of time horizon, reactor-types and characteristics, capacities
at the beginning */
			/* and total capacities for each class after each period
*/
			input_cap(infile_cap, &classes, &TotalClassCap, &NewTypeCap,
&TotalTypeCap, &Production);


			if(error == 0)
			{
				/* interacitve function for input of capacities for each type
within each period */
```

```
                        new_capacity(infile_cap, &classes, &TotalClassCap, &NewTypeCap,
&TotalTypeCap, &Production);

                        if(error == 0)
                                output_cap(outfile_cap, &classes, &TotalClassCap,
&NewTypeCap, &TotalTypeCap, &Production);
                }
        }


        fclose(infile_cap);
        fclose(outfile_cap);

        /* end part I */

        if(error == 0)
        {
                /* part II */


                /*  ascertaining of total number of types */
                ttypes = classes[RESID].number_of_types + classes[STR].number_of_types +
classes[PW].number_of_types;

                /*  memory allocation of all used variables  */
                memory_allocation_mass(ttypes, &Matrix, &Matin, &TMatin1, &TMatin2,
&TMatout1, &TMatout2,
                                                        &TMatout, &Storage, &Refab,
&Unat, &Unatt, &Tht, &UirrWaste,
                                                        &PuWaste, &MaWaste, &Waste);

                if(error == 0)
                        /*  data input  */
                        input_mass(infile_mass1, infile_mass2, ttypes, &Matrix,
&Storage, &StorageResid,
                                                &UirrWaste, &PuWaste, &MaWaste,
&UirrWasteResid, &PuWasteResid, &MaWasteResid,
                                                &VPU, &VUIRR, &VMA);


                if(error == 0)
                {
                        /* calculation of incoming substances */
                        sub_in(ttypes, &Matrix, &Matin, &TMatin1, &TMatin2, &Refab,
&Unat, &Unatt, &Tht, &Production, &NewTypeCap, &classes);

                        if(error == 0)
                        {
                                /* calculation of outcoming substances */
                                sub_out(ttypes, &Matrix, &Matin, &TMatout1, &TMatout2,
&TMatout, &Production, &NewTypeCap, &Storage, &classes);

                                if(error == 0)
                                {
                                        /* calculation of waste */
                                        waste_calc(ttypes, &Matin, &Storage, &UirrWaste,
&PuWaste, &MaWaste, &Waste, VPU, VUIRR, VMA);

                                        if(error == 0)
                                                output_mass(outfile_mass, &classes,
ttypes, &Matin, &TMatin1, &TMatin2, &TMatout, &Storage,
                                                                        &Refab, &Unat,
&Unatt, &Tht, &StorageResid, &UirrWaste, UirrWasteResid,
                                                                        &PuWaste,
PuWasteResid, &MaWaste, MaWasteResid, &Waste);
                                }
                        }


                }

                fclose(infile_mass1);
                fclose(infile_mass2);
                fclose(outfile_mass);
        }

    }
```

```
	if(error!=0)
		error_output();
	else
		mass_results(ttypes, &classes, &TMatin1, &TMatin2, &TMatout1, &TMatout2,
&StorageResid, UirrWasteResid, PuWasteResid, MaWasteResid, VPU, VUIRR, VMA);

	/* create charts */
	system("create_charts.exe");

	return 1;
}
```

/* filename: vleem2_capacity_memory_allocation.c */

```
#include "vleem2_capacity_header.h"

void memory_allocation_cap(reactor_class** classes, float*** TotalClassCap,
                  float**** NewTypeCap, float**** TotalTypeCap, float**** Production)
{

       int i,j;

       /*  memory allocation for classes[j]  */
       for(j=0; j<3; j++)
       {
              assert((*classes)[j].types = (reactor_type*)
malloc((*classes)[j].number_of_types*sizeof(reactor_type)));
              for(i=0; i<(*classes)[j].number_of_types; i++)
                     assert((*classes)[j].types[i].name = (char*)malloc(20*sizeof(char)));
       }


       /*  memory allocation for TotalClassCap  */
       assert((*TotalClassCap) = (float**)malloc((IT+1)*sizeof(float*)));
       for(i=0; i<=IT; i++)
              assert((*TotalClassCap)[i] = (float*)malloc(3*sizeof(float)));


       /*  memory allocation for TotalTypeCap  */
       assert((*TotalTypeCap) = (float***) malloc((IT+1)*sizeof(float**)));
       for(i=0; i<=IT; i++)
       {
              assert((*TotalTypeCap)[i] = (float**) malloc(3*sizeof(float*)));
              for(j=0; j<3; j++)
                     assert((*TotalTypeCap)[i][j] = (float*)
malloc((*classes)[j].number_of_types*sizeof(float)));
       }


       /*  memory allocation for NewTypeCap  */
       assert((*NewTypeCap) = (float***) malloc(IT*sizeof(float**)));
       for(i=0; i<IT; i++)
       {
              assert((*NewTypeCap)[i] = (float**) malloc(3*sizeof(float*)));
              for(j=0; j<3; j++)
                     assert((*NewTypeCap)[i][j] = (float*)
malloc((*classes)[j].number_of_types*sizeof(float)));
       }


       /*  memory allocation for Production  */
       assert((*Production) = (float***) malloc(IT*sizeof(float**)));
       for(i=0; i<IT; i++)
       {
              assert((*Production)[i] = (float**) malloc(3*sizeof(float*)));
              for(j=0; j<3; j++)
                     assert((*Production)[i][j] = (float*)
malloc((*classes)[j].number_of_types*sizeof(float)));
       }

}
```

/* filename: vleem2_capacity_input.c */

```
#include "vleem2_capacity_header.h"

/* maximum time of full load  */
#define MAX_FLH 8760

void input_cap(FILE* infile, reactor_class** classes, float*** TotalClassCap,
                  float**** NewTypeCap, float**** TotalTypeCap, float**** Production)
{
       int i,j,k;
       char* helpline;
       int character;

       assert(helpline= (char*)malloc(400*sizeof(char)));

       /*  input time horizon  */
       if(fscanf(infile, "%[^0123456789]%d", helpline, &JI)!=2)
              error = INPUT_TH_ERROR;
       else if((JI<1900) || (JI>3000))
              error = INPUT_TH_ERROR;
       else
       {
              if(fscanf(infile, "%[^0123456789]%d", helpline, &JF)!=2)
                     error = INPUT_TH_ERROR;
              else if((JF<=JI) || (JF<1900) || (JF>3000))
                     error = INPUT_TH_ERROR;
              else
              {
                     if(fscanf(infile, "%[^0123456789]%d", helpline, &JT)!=2)
                            error = INPUT_TH_ERROR;
                     else if((JT<0) || (((JF-JI)%JT)!=0))
                            error = INPUT_TH_ERROR;
              }
       }

       /*  number of intervals  */
       if(error==0)
              IT = (JF-JI)/JT;


       /*  input of reactor-types for each reactor-class  */
       for(j=0; j<3; j++)
       {
              if(fscanf(infile, "%[^0123456789]%d", helpline,
&((*classes)[j].number_of_types))!=2)
                     error = INPUT_TYPENO_ERROR;
              else if((*classes)[j].number_of_types < 0)
                     error = INPUT_TYPENO_ERROR;
       }


       if(error==0)
       {
              /*  allocation for all global variables  */
              memory_allocation_cap(classes, TotalClassCap, NewTypeCap, TotalTypeCap,
Production);

              if(error==0)
              {

                     for(j=0; j<3; j++)
                     {

                            /* space */
                            while(isspace(character = fgetc(infile)));
                            ungetc(character, infile);
                            /* scanf of comment-line in input */
                            if(fgets(helpline, 400, infile)==NULL)
                                   error = INPUT_FORMAT_ERROR;



                            /*  classes[j]  */
                            for(i=0; (i<(*classes)[j].number_of_types) && (error==0); i++)
                            {
                                   (*classes)[j].types[i].nr = i;
```

```
                                        if(fscanf(infile, "%s", (*classes)[j].types[i].name)!=1)
                                                error = INPUT_NAME_ERROR;

                                        if(fscanf(infile, "%d",
&((*classes)[j].types[i].lifetime))!=1)
                                                error = INPUT_LT_ERROR;

                                        if(fscanf(infile, "%f",
&((*classes)[j].types[i].full_load_hour))!=1)
                                                error = INPUT_FLH_ERROR;
                                        (*classes)[j].types[i].full_load_hour /= 1000;

                                        if(fscanf(infile, "%d",
&((*classes)[j].types[i].cooling_time))!=1)
                                                error = INPUT_CT_ERROR;

                                        if((((*classes)[j].types[i].lifetime % JT)!=0) ||
                                                (((*classes)[j].types[i].full_load_hour * 1000) >
MAX_FLH))

                                                error = INPUT_LT_FLH_ERROR;
                                }
                        }
                }
        }

        /* space */
        while(isspace(character = fgetc(infile)));
        ungetc(character, infile);
        /* scanf of comment-line in input */
        if(fgets(helpline, 400, infile)==NULL)
                error = INPUT_FORMAT_ERROR;


        /*  input of TotalTypeCap at the beginning of the first interval  */
        for(j=0; j<3; j++)
        {

                for(k=0; k<(*classes)[j].number_of_types; k++)
                {
                        if(fscanf(infile, "%f", &((*TotalTypeCap)[0][j][k]))!=1)
                                error = INPUT_TTC_ERROR;
                }
        }



        if(!error)
        {
                /*  total capacity for each class at the beginning of the first interval  */
                for(j=0; j<3; j++)
                {
                        (*TotalClassCap)[0][j] = 0;
                        for(k=0; k<(*classes)[j].number_of_types; k++)
                                (*TotalClassCap)[0][j] += (*TotalTypeCap)[0][j][k];
                }



                /* space */
                while(isspace(character = fgetc(infile)));
                ungetc(character, infile);
                /* scanf of comment-line in input */
                if(fgets(helpline, 400, infile)==NULL)
                        error = INPUT_FORMAT_ERROR;


                /*  input of total capacity for each interval for each class */
                for(j=0; j<3; j++)
                {

                        if(fscanf(infile, "%[^01234567890]", helpline)!=1)
                                error = INPUT_FORMAT_ERROR;

                        for(i=1; (i<=IT) && (error==0); i++)
                        {

                                if(fscanf(infile, "%f", &((*TotalClassCap)[i][j]))!=1)
```

```
                                error = INPUT_TCC_ERROR;
                        else if((*TotalClassCap)[i][j] <0)
                                error = INPUT_TCC_ERROR;
                        }
                }

                /* space */
                while(isspace(character = fgetc(infile)));
                ungetc(character, infile);
        }
}
```

/* filename: vleem2_capacity_new_cap.c */

```
#include "vleem2_capacity_header.h"

void new_capacity(FILE* infile, reactor_class** classes, float*** TotalClassCap,
                  float**** NewTypeCap, float**** TotalTypeCap, float**** Production)
{
       int i,j,k,l, character;
       float type_cap;
       float cap_needed;
       char* helpline;


       assert(helpline= (char*)malloc(400*sizeof(char)));

       /* scanf of comment-line in input */
       if(fgets(helpline, 400, infile)==NULL)
              error = INPUT_FORMAT_ERROR;
       /* space */
       while(isspace(character = fgetc(infile)));
       ungetc(character, infile);

       for(j=0; j<3; j++)
       /*  input for each class  */
       {
              for(k=0; k<(*classes)[j].number_of_types; k++)
              /* input for each type */
              {
                     /* scanf of comment-line in input */
                     while(!isdigit(character = fgetc(infile)));
                     ungetc(character, infile);

                     for(i=1; i<=IT; i++)
                     /* input for each interval */
                     {
                            if(fscanf(infile, "%f", &((*NewTypeCap)[i-1][j][k]))!=1)
                                   error = INPUT_NEWCAP_ERROR;
                            else if((*NewTypeCap)[i-1][j][k]<0)
                                   error = INPUT_NEWCAP_ERROR;
                     }
              }
       }

       if(error == 0)
       {
              /* check of input data */
              for(i=1; i<=IT; i++)
              {
                     for(j=0; j<3; j++)
                     {
                            for(k=0; k<(*classes)[j].number_of_types; k++)
                            {
                                   if(error == 0)
                                   {
                                          if(j==0)
                                          {
                                                 type_cap = (*TotalTypeCap)[0][0][k];
                                                 for(l=0; l<=i-1; l++)
                                                        type_cap -= (*NewTypeCap)[l][0][k];

                                                 if(type_cap < 0)
                                                 {
                                                        error = INPUT_NEG_RESID_ERROR;
                                                        error_int = i;
                                                        error_class = j;
                                                 }
                                          }

                                          if(j==0)
                                                 cap_needed = (*TotalClassCap)[i-1][j]-
(*TotalClassCap)[i][j];
                                          else
                                          {
                                                 cap_needed = (*TotalClassCap)[i][j]-
(*TotalClassCap)[i-1][j];
                                                 for(k=0; k<(*classes)[j].number_of_types;
k++)
```

```
                                                        cap_needed +=
still(&((*classes)[j]), &((*classes)[j].types[k]), i, NewTypeCap);
                                                }

                                                if(difference = diff(&((*classes)[j]),i,
TotalClassCap, NewTypeCap))
                                                {
                                                        if(difference > 0)
                                                        {
                                                                error = INPUT_POSDIFF_ERROR;
                                                                error_int = i;
                                                                error_class = j;
                                                        }
                                                        else if(difference < 0)
                                                        {
                                                                error = INPUT_NEGDIFF_ERROR;
                                                                error_int = i;
                                                                error_class = j;
                                                        }
                                                }
                                        }
                                }
                        }
                }
        }

        if(error == 0)
        {
                for(i=1; i<=IT; i++)
                {
                        /*  difference of each class = 0  -> calculate nuclear individual
capacity for each type */
                        for(j=0; j<3; j++)
                        {
                                for(k=0; k<(*classes)[j].number_of_types; k++)
                                        (*TotalTypeCap)[i][j][k] = cap(&((*classes)[j]),
&((*classes)[j].types[k]), i, NewTypeCap, TotalTypeCap);
                        }
                        /*  production  */
                        for(j=0; j<3; j++)
                        {
                                for(k=0; k<(*classes)[j].number_of_types; k++)
                                        (*Production)[i-1][j][k] = pro(&((*classes)[j]),
&((*classes)[j].types[k]), i, NewTypeCap, TotalTypeCap);
                        }
                }
        }
}
```

/* filename: vleem2_capacity_calculation.c */

```
#include "vleem2_capacity_header.h"

float tcap(reactor_class* r_class, int interval, float*** TotalClassCap)
{

	/*  total capacity of one class after(!) one interval  */
	if(interval<0)
		return 0;
	else
		return ((*TotalClassCap)[interval][r_class->nr]);

}

float cap_new(reactor_class* r_class, reactor_type* r_type, int interval, float****
NewTypeCap)
{

	if((interval<=0) || (r_class->nr == RESID))
	/*  at the beginning  or RESID-class  */
		return 0;
	else
		return ((*NewTypeCap)[interval-1][r_class->nr][r_type->nr]);

}

float still(reactor_class* r_class, reactor_type* r_type, int interval, float**** NewTypeCap)
{

	if(interval<=0)
		return 0;
	else
	{
		if(r_class->nr == RESID)
			return ((*NewTypeCap)[interval-1][RESID][r_type->nr]);

		else
			return (cap_new(r_class, r_type, interval- ((r_type->lifetime)/JT),
NewTypeCap));
	}

}

float tcap_test(reactor_class* r_class, int interval, float*** TotalClassCap, float****
NewTypeCap)
{

	int i;
	float total_cap_new=0, total_still=0;

	for(i=0; i<r_class->number_of_types; i++)
	{
		total_cap_new += cap_new(r_class, &(r_class->types[i]), interval, NewTypeCap);
		total_still += still(r_class, &(r_class->types[i]), interval, NewTypeCap);
	}

	return (tcap(r_class, interval-1, TotalClassCap) + total_cap_new - total_still);

}

float diff(reactor_class* r_class, int interval, float*** TotalClassCap, float**** NewTypeCap)
{

	return (tcap_test(r_class, interval, TotalClassCap, NewTypeCap) - tcap(r_class,
interval, TotalClassCap));

}

float cap(reactor_class* r_class, reactor_type* r_type, int interval, float**** NewTypeCap,
float**** TotalTypeCap)
{
	/*  condition of breaking off  */
	if(interval==0)
		return ((*TotalTypeCap)[interval][r_class->nr][r_type->nr]);
	return (cap(r_class, r_type, interval-1, NewTypeCap, TotalTypeCap) + cap_new(r_class,
r_type, interval, NewTypeCap) - still(r_class, r_type, interval, NewTypeCap));
```

```
}

float pro(reactor_class* r_class, reactor_type* r_type, int interval, float**** NewTypeCap,
float**** TotalTypeCap)
{

        return (cap(r_class, r_type, interval, NewTypeCap, TotalTypeCap) * r_type-
>full_load_hour * JT);

}
```

/* filename: vleem2_capacity_output.c */

```
#include "vleem2_capacity_header.h"



void dezkomma(char* str, float wert)
{
	int index=0;

	sprintf(str, "%.1f", wert);
	while(str[index] != '\0')
	{
		if(str[index] == '.')
		{
			str[index] = ',';
			break;
		}
		index++;
	}

}


void output_cap(FILE* outfile, reactor_class** classes, float*** TotalClassCap,
		  float**** NewTypeCap, float**** TotalTypeCap, float**** Production)
{

	int i,j,k;
	float reached_cap;
	char hilfsstring[20];

	/*  print intervals in columns and types in rows  */

	fprintf(outfile, "%s  \t", "Interval");
	for(i=0; i<=IT; i++)
	{
		if(i==0)
			fprintf(outfile, "%d \t", JI);
		else
			fprintf(outfile, "%4d-%4d \t", JI+(i-1)*JT, JI+i*JT);
	}

	fprintf(outfile, "\n\n");
	for(j=0; j<3; j++)
	{
		for(k=0; k<(*classes)[j].number_of_types; k++)
		{
			/*  total capacity in each in interval */
			fprintf(outfile, "%s tot.cap. \t", (*classes)[j].types[k].name);
			for(i=0; i<=IT; i++)
			{
				dezkomma(hilfsstring, (*TotalTypeCap)[i][j][k]);
				fprintf(outfile, "%s \t", hilfsstring);
			}
			fprintf(outfile, "\n");

			/*  new capacity/ closure  */
			if(j==0)
				fprintf(outfile, "%s closed   \t", (*classes)[j].types[k].name);
			else
				fprintf(outfile, "%s new cap. \t", (*classes)[j].types[k].name);
			for(i=0; i<=IT; i++)
			{
				if(i==0)
					fprintf(outfile, "\t");
				else
				{
					dezkomma(hilfsstring, (*NewTypeCap)[i-1][j][k]);
					fprintf(outfile, "%s \t", hilfsstring);
				}
			}
			fprintf(outfile, "\n");

			/*  production  */
			fprintf(outfile, "%s prod.    \t", (*classes)[j].types[k].name);
			for(i=0; i<=IT; i++)
			{
```

```
                        if(i==0)
                                fprintf(outfile, "\t");
                        else
                        {
                                dezkomma(hilfsstring, (*Production)[i-1][j][k]);
                                fprintf(outfile, "%s \t", hilfsstring);
                        }
                }
                fprintf(outfile, "\n");

        }

        fprintf(outfile, "\n\n");

        /* Total capacity planned for each class */
        fprintf(outfile, "%s planned  \t", (*classes)[j].name);
        for(i=0; i<=IT; i++)
        {
                dezkomma(hilfsstring, (*TotalClassCap)[i][j]);
                fprintf(outfile, "%s \t",hilfsstring);
        }
        fprintf(outfile, "\n");

        /* Total capacity for each class */
        fprintf(outfile, "%s reached  \t", (*classes)[j].name);
        for(i=0; i<=IT; i++)
        {
                reached_cap = 0;
                for(k=0; k<(*classes)[j].number_of_types; k++)
                        reached_cap += (*TotalTypeCap)[i][j][k];

                dezkomma(hilfsstring, reached_cap);
                fprintf(outfile, "%s \t", hilfsstring);
        }
        fprintf(outfile, "\n\n");
    }
}
```

/* filename: vleem2_mass_memory_allocation.c */

```
#include "vleem2_capacity_header.h"
#include "vleem2_mass_header.h"

void memory_allocation_mass(int ttypes, float*** Matrix,float*** Matin, float**** TMatin1,
float****TMatin2,
                                                    float**** TMatout1,   float**** TMatout2
,float**** TMatout,
                                                    float*** Storage, float** Refab, float**
Unat, float** Unatt, float** Tht,
                                                    float** UirrWaste, float** PuWaste,
float** MaWaste, float** Waste)
{

        int i,j;

        /*  memory allocation for Matrix  */
        assert((*Matrix) = (float**) malloc((3*ttypes)*sizeof(float*)));
        for(i=0; i<3*ttypes; i++)
                assert((*Matrix)[i] = (float*) malloc(16*sizeof(float)));


        /*  memory allocation for Matin  */
        assert((*Matin) = (float**) malloc(IT*sizeof(float*)));
        for(i=0; i<IT; i++)
                assert((*Matin)[i] = (float*) malloc(9*sizeof(float)));



        /*  memory allocation for TMatin1  */
        assert((*TMatin1) = (float***) malloc(IT*sizeof(float**)));
        for(i=0; i<IT; i++)
        {
                assert((*TMatin1)[i] = (float**) malloc(9*sizeof(float*)));

                for(j=0; j<9; j++)
                        assert((*TMatin1)[i][j] = (float*) malloc(ttypes*sizeof(float)));
        }

        /*  memory allocation for TMatin2  */
        assert((*TMatin2) = (float***) malloc(IT*sizeof(float**)));
        for(i=0; i<IT; i++)
        {
                assert((*TMatin2)[i] = (float**) malloc(9*sizeof(float*)));

                for(j=0; j<9; j++)
                        assert((*TMatin2)[i][j] = (float*) malloc(ttypes*sizeof(float)));
        }


        /*  memory allocation for TMatout1  */
        assert((*TMatout1) = (float***) malloc(IT*sizeof(float**)));
        for(i=0; i<IT; i++)
        {
                assert((*TMatout1)[i] = (float**) malloc(7*sizeof(float*)));

                for(j=0; j<7; j++)
                        assert((*TMatout1)[i][j] = (float*) malloc(ttypes*sizeof(float)));
        }


        /*  memory allocation for TMatout2  */
        assert((*TMatout2) = (float***) malloc(IT*sizeof(float**)));
        for(i=0; i<IT; i++)
        {
                assert((*TMatout2)[i] = (float**) malloc(7*sizeof(float*)));

                for(j=0; j<7; j++)
                        assert((*TMatout2)[i][j] = (float*) malloc(ttypes*sizeof(float)));
        }


        /*  memory allocation for TMatout  */
        assert((*TMatout) = (float***) malloc(IT*sizeof(float**)));
        for(i=0; i<IT; i++)
        {
                assert((*TMatout)[i] = (float**) malloc(7*sizeof(float*)));
```

```
                for(j=0; j<7; j++)
                        assert((*TMatout)[i][j] = (float*) malloc(ttypes*sizeof(float)));
        }

        /*  memory allocation for Storage  */
        assert((*Storage) = (float**) malloc(IT*sizeof(float*)));
        for(i=0; i<IT; i++)
                assert((*Storage)[i] = (float*) malloc(7*sizeof(float)));

        /*  memory allocation for Refab  */
        assert((*Refab) = (float*) malloc(IT * sizeof(float)));

        /*  memory allocation for Unat  */
        assert((*Unat) = (float*) malloc(IT * sizeof(float)));

        /*  memory allocation for Unatt  */
        assert((*Unatt) = (float*) malloc(IT * sizeof(float)));

        /*  memory allocation for Tht */
        assert((*Tht) = (float*) malloc(IT * sizeof(float)));

        /*  memory allocation for UirrWaste  */
        assert((*UirrWaste) = (float*) malloc(IT * sizeof(float)));

        /*  memory allocation for PuWaste  */
        assert((*PuWaste) = (float*) malloc(IT * sizeof(float)));

        /*  memory allocation for MaWaste  */
        assert((*MaWaste) = (float*) malloc(IT * sizeof(float)));

        /*  memory allocation for Waste  */
        assert((*Waste) = (float*) malloc(IT * sizeof(float)));

}
```

/* filename: vleem2_mass_input.c */

```
#include "vleem2_capacity_header.h"
#include "vleem2_mass_header.h"

/*  number of substances  */
#define NMAT 16

void input_mass(FILE* infile_mass1, FILE* infile_mass2, int ttypes, float*** Matrix,
                              float*** Storage, float** StorageResid, float** UirrWaste,
float** PuWaste, float** MaWaste,
                              float* UirrWasteResid, float* PuWasteResid, float* MaWasteResid,
                              float* VPU, float* VUIRR, float* VMA)
{

        int i,j;
        substance sub;
        char* helpline;
        int character;


        assert(helpline= (char*)malloc(400*sizeof(char)));
        /* scanf of 2 comment-lines in input */
        if(fgets(helpline, 400, infile_mass1)==NULL)
                error = INPUT_FORMAT_ERROR;
        if(fgets(helpline, 400, infile_mass1)==NULL)
                        error = INPUT_FORMAT_ERROR;


        /* input of Matrix */
        for(i=0; i<3; i++)
        /* 3* ttypes */
        {
                /* scanf of comment-line in input */
                if(fgets(helpline, 400, infile_mass1)==NULL)
                        error = INPUT_FORMAT_ERROR;

                for(j=0; j<ttypes; j++)
                {
                        /* input of type_name */
                        if(fscanf(infile_mass1, "%[^;]", helpline)!=1)
                                error = INPUT_FORMAT_ERROR;


                        sub = UnatIn;
                        while(sub<PurecOut)
                        {
                                character = fgetc(infile_mass1);
                                if((character = fgetc(infile_mass1)) == ';')
                                {
                                        /* empty cell */
                                        ungetc(character, infile_mass1);
                                        (*Matrix)[i*ttypes+j][sub-1] = 0;
                                }
                                else
                                {
                                        ungetc(character, infile_mass1);
                                        if(fscanf(infile_mass1, "%f",
&((*Matrix)[i*ttypes+j][sub-1])) != 1)
                                                error = INPUT_MATRIX_ERROR;
                                }

                                sub++;
                        }
                        /* sub = PurecOut */
                        character = fgetc(infile_mass1);
                        if((character = fgetc(infile_mass1)) == '\n')
                        {
                                /* empty cell */
                                (*Matrix)[i*ttypes+j][sub-1] = 0;
                        }
                        else
                        {
                                ungetc(character, infile_mass1);
                                if(fscanf(infile_mass1, "%f", &((*Matrix)[i*ttypes+j][sub-1]))
!= 1)
                                        error = INPUT_MATRIX_ERROR;
                        }
```

```
                /* rest of line */
                while(isspace(character = fgetc(infile_mass1)));
                ungetc(character, infile_mass1);
        }
}

/*  initial value for storage  */

assert((*StorageResid) = (float*) malloc(7* sizeof(float)));

/* scanf of comment-line in input */
while(!isdigit(character = fgetc(infile_mass2)));
ungetc(character, infile_mass2);

for(sub=ThUOut; sub<PurecOut; sub++)
{
        if(fscanf(infile_mass2, "%f", &((*Storage)[0][sub-ThUOut])) != 1)
                error = INPUT_STORAGE_ERROR;

        if(error == 0)
                (*StorageResid)[sub-ThUOut] = (*Storage)[0][sub-ThUOut];

        if((character = fgetc(infile_mass2)) != ';')
                error = INPUT_FORMAT_ERROR;
}
/* sub == PurecOut */
if(fscanf(infile_mass2, "%f", &((*Storage)[0][sub-ThUOut])) != 1)
        error = INPUT_STORAGE_ERROR;
if (error == 0)
        (*StorageResid)[sub-ThUOut] = (*Storage)[0][sub-ThUOut];

/*  initial value for waste  */
/* scanf of comment-line in input */
while(!isdigit(character = fgetc(infile_mass2)));
ungetc(character, infile_mass2);

if(fscanf(infile_mass2, "%f", &((*UirrWaste)[0])) != 1)
        error = INPUT_WASTE_ERROR;
if(error == 0)
        *UirrWasteResid = (*UirrWaste)[0];
if((character = fgetc(infile_mass2)) != ';')
        error = INPUT_FORMAT_ERROR;

if(fscanf(infile_mass2, "%f", &((*PuWaste)[0])) != 1)
        error = INPUT_WASTE_ERROR;
if(error==0)
        *PuWasteResid = (*PuWaste)[0];
if((character = fgetc(infile_mass2)) != ';')
        error = INPUT_FORMAT_ERROR;

if(fscanf(infile_mass2, "%f", &((*MaWaste)[0])) != 1)
        error = INPUT_WASTE_ERROR;
if(error==0)
        *MaWasteResid = (*MaWaste)[0];


/*  loss factors  */

while(!isdigit(character = fgetc(infile_mass2)));
ungetc(character, infile_mass2);

/* VPU */
if(fscanf(infile_mass2, "%f", VPU) != 1)
        error = INPUT_FACTOR_ERROR;

while(!isdigit(character = fgetc(infile_mass2)));
ungetc(character, infile_mass2);

/* VUIRR */
if(fscanf(infile_mass2, "%f", VUIRR) != 1)
        error = INPUT_FACTOR_ERROR;

while(!isdigit(character = fgetc(infile_mass2)));
ungetc(character, infile_mass2);

/* VMA */
if(fscanf(infile_mass2, "%f", VMA) != 1)
```

```
                error = INPUT_FACTOR_ERROR;
}
```

/* filename: vleem2_mass_substance_calculation */

```
#include "vleem2_capacity_header.h"
#include "vleem2_mass_header.h"

/* calculation of incoming substances */
void sub_in(int ttypes, float*** Matrix, float*** Matin, float**** TMatin1, float**** TMatin2
,float** Refab, float** Unat, float** Unatt,
                    float** Tht, float**** Production, float**** NewTypeCap, reactor_class**
classes)
{

       int i, j, k, type;
       substance mat;

       for(i=1; i<=IT; i++)
       {
              for(mat=UnatIn; mat<=HmIn; mat++)
              {
                     (*Matin)[i-1][mat-1] = 0;
                     type = 1;
                     for(j=RESID; j<=PW; j++)
                     {
                            for(k=0; k<(*classes)[j].number_of_types; k++)
                            {
                                   (*TMatin1)[i-1][mat-1][type-1] = (*Production)[i-
1][j][k]* (*Matrix)[type-1][mat-1];
                                   (*TMatin2)[i-1][mat-1][type-1] =
cap_new(&((*classes)[j]), &((*classes)[j].types[k]),i,NewTypeCap) * (*Matrix)[ttypes+type-
1][mat-1];

                                   (*Matin)[i-1][mat-1] += (*TMatin1)[i-1][mat-1][type-1] +
(*TMatin2)[i-1][mat-1][type-1];
                                   type++;
                            }
                     }
              }

              (*Refab)[i-1] = (*Matin)[i-1][HmIn-1];
              (*Unat)[i-1] = (*Matin)[i-1][UnatIn-1];
       }

       (*Unatt)[0] = (*Unat)[0];
       (*Tht)[0] = (*Matin)[0][ThIn-1];

       for(i=2; i<=IT; i++)
       {
              (*Unatt)[i-1] = (*Unatt)[i-2] + (*Unat)[i-1];
              (*Tht)[i-1] = (*Tht)[i-2] + (*Matin)[i-1][ThIn-1];
       }
}



/* calculation of outcoming substances */
void sub_out(int ttypes, float*** Matrix, float*** Matin, float**** TMatout1, float****
TMatout2, float**** TMatout,
                     float**** Production, float**** NewTypeCap, float*** Storage,
reactor_class** classes)
{
       int i, j, k, type;
       substance mat;
       float factor;

       for(i=1; i<=IT; i++)
       {
              for(mat=ThUOut; mat<=PurecOut; mat++)
              {
                     type = 1;
                     for(j=RESID; j<=PW; j++)
                     {
                            for(k=0; k<(*classes)[j].number_of_types; k++)
                            {
                                   (*TMatout1)[i-1][mat-10][type-1] = (*Production)[i-
1][j][k]* (*Matrix)[type-1][mat-1];
                                   (*TMatout2)[i-1][mat-10][type-1] =
still(&((*classes)[j]), &((*classes)[j].types[k]),i, NewTypeCap)
```

```
                * (*Matrix)[2*ttypes + type-1][mat-1];

                                        (*TMatout)[i-1][mat-10][type-1] = (*TMatout1)[i-1][mat-
10][type-1] + (*TMatout2)[i-1][mat-10][type-1];
                                        type++;
                                }
                        }
                }
        }


        for(i=1; i<=IT; i++)
        {
                for(mat=ThUOut; mat<=PufOut; mat++)
                {
                        factor = 0;
                        type = 1;
                        for(j=RESID; j<=PW; j++)
                        {
                                for(k=0; k<(*classes)[j].number_of_types; k++)
                                {
                                        if((i-(*classes)[j].types[k].cooling_time) > 0)
                                                factor += (*TMatout)[i-
(*classes)[j].types[k].cooling_time -1][mat-10][type-1];
                                        type++;
                                }
                        }

                        if(i==1)
                                (*Storage)[0][mat-10] += factor;
                        else
                                (*Storage)[i-1][mat-10] = (*Storage)[i-2][mat-10] + factor;

                        if(mat>ThUOut)
                                (*Storage)[i-1][mat-10] -= (*Matin)[i-1][mat-9];
                }

                /* mat==PurecOut */
                if(i==1)
                        (*Storage)[0][PurecOut-10] += factor - (*Matin)[0][PufIn-1];
                else
                        (*Storage)[i-1][PurecOut-10] = (*Storage)[i-2][PurecOut-10] + factor -
(*Matin)[i-1][PufIn-1];

                factor = 0;
                type = 1;
                for(j=RESID; j<=PW; j++)
                {
                        for(k=0; k<(*classes)[j].number_of_types; k++)
                        {
                                if((i-(*classes)[j].types[k].cooling_time) > 0)
                                        factor += (*TMatout)[i-
(*classes)[j].types[k].cooling_time -1][PurecOut-10][type-1];
                                type++;
                        }
                }

                (*Storage)[i-1][PurecOut-10] += factor - (*Matin)[i-1][PurecIn-1];
        }
}



/*  calculation of waste  */
void waste_calc(int ttypes, float*** Matin, float*** Storage, float** UirrWaste, float**
PuWaste,
                                float** MaWaste, float** Waste, float VPU, float VUIRR, float
VMA)
{
        int i;

        for(i=1; i<=IT; i++)
        {
                if(i==1)
                {
                        (*UirrWaste)[0] += VUIRR * ((*Matin)[0][UirrIn-1]);
```

```
                        (*PuWaste)[0] += VPU * ((*Matin)[0][PufIn-1] + (*Matin)[0][PurecIn-1]);
                        (*MaWaste)[0] += VMA * (*Matin)[0][MaIn-1];
                }
                else
                {
                        (*UirrWaste)[i-1] = (*UirrWaste)[i-2] + VUIRR * (*Matin)[i-1][UirrIn-1];
                        (*PuWaste)[i-1] = (*PuWaste)[i-2] + VPU * ((*Matin)[i-1][PufIn-1] +
(*Matin)[i-1][PurecIn-1]);
                        (*MaWaste)[i-1] = (*MaWaste)[i-2] + VMA * (*Matin)[i-1][MaIn-1];
                }

                (*Waste)[i-1] = (*Storage)[i-1][FpOut-1] + (*UirrWaste)[i-1] + (*PuWaste)[i-1]
+ (*MaWaste)[i-1];
        }
}
```

/* filename: vleem2_mass_output.c */

```
#include "vleem2_capacity_header.h"
#include "vleem2_mass_header.h"
#include "vleem2_header.h"


void output_mass(FILE* outfile, reactor_class** classes, int ttypes, float*** Matin, float****
TMatin1, float**** TMatin2, float**** TMatout, float*** Storage,
                              float** Refab, float** Unat, float** Unatt, float** Tht,
float** StorageResid, float** UirrWaste, float UirrWasteResid, float** PuWaste,
                              float PuWasteResid, float** MaWaste, float MaWasteResid,
float** Waste)

{
        int interval,type, i,j,k;
        substance mat;
        char *matname[] = { "UnatIn","ThIn","FpIn","UdeplIn","UirrIn","MaIn","PufIn","PurecIn",

        "HmIn","ThUOut","FpOut","UdeplOut","UirrOut","MaOut","PufOut","PurecOut"};
        char **typenames;
        char hilfsstring[20];

        /* typenames */
        assert(typenames = (char**) malloc(ttypes * sizeof(char*)));
        for(i=0; i<ttypes; i++)
                typenames[i] = (char*) malloc(20 * sizeof(char));

        i=0;
        for(j=RESID; j<=PW; j++)
        {
                for(k=0; k<(*classes)[j].number_of_types; k++)
                {
                        strcpy(typenames[i], (*classes)[j].types[k].name);
                        i++;
                }
        }



        /* column headlines */
        fprintf(outfile, "%s  \t", "Interval");
        for(interval=0; interval<=IT; interval++)
        {
                if(interval==0)
                        fprintf(outfile, "%s \t", "RESID");
                else
                        fprintf(outfile, "%4d-%4d \t", JI+(interval-1)*JT, JI+interval*JT);
        }
        fprintf(outfile, "\n\n");

        /* output of TMatin1, TMatin2 & Matin */
        for(mat=UnatIn; mat<=HmIn; mat++)
        {
                for(type=1; type<=ttypes; type++)
                {
                        fprintf(outfile, "%s from %s \t", matname[mat-1], typenames[type-1]);
                        for(interval=0; interval<=IT; interval++)
                        {
                                if(interval==0)
                                        fprintf(outfile, "\t");
                                else
                                {
                                        dezkomma(hilfsstring, (*TMatin1)[interval-1][mat-1][type-
1] + (*TMatin2)[interval-1][mat-1][type-1]);
                                        fprintf(outfile, "%s \t", hilfsstring);
                                }
                        }
                        fprintf(outfile, "\n");
                }
                fprintf(outfile, "\n");
                fprintf(outfile, "%s  \t", matname[mat-1]);
                for(interval=0; interval<=IT; interval++)
                        {
                        if(interval==0)
                                fprintf(outfile, "\t");
                        else
                        {
```

```
                                dezkomma(hilfsstring, (*Matin)[interval-1][mat-1]);
                                fprintf(outfile, "%s \t", hilfsstring);
                        }
                }
                fprintf(outfile, "\n\n");
        }
        fprintf(outfile, "\n");

        /* output of TMatout */
        for(mat=ThUOut; mat<=PurecOut; mat++)
        {
                for(type=1; type<=ttypes; type++)
                {
                        fprintf(outfile, "%s from %s \t", matname[mat-1], typenames[type-1]);
                        for(interval=0; interval<=IT; interval++)
                        {
                                if(interval==0)
                                        fprintf(outfile, "\t");
                                else
                                {
                                        dezkomma(hilfsstring, (*TMatout)[interval-1][mat-
10][type-1]);
                                        fprintf(outfile, "%s \t", hilfsstring);
                                }
                        }
                        fprintf(outfile, "\n");
                }
                fprintf(outfile, "\n");
        }
        fprintf(outfile, "\n");

        /* output of Storage */
        for(mat=ThUOut; mat<=PurecOut; mat++)
        {
                fprintf(outfile, "%s storage \t", matname[mat-1]);
                for(interval=0; interval<=IT; interval++)
                {
                        if(interval==0)
                        {
                                dezkomma(hilfsstring, (*StorageResid)[mat-10]);
                                fprintf(outfile, "%s \t", hilfsstring);
                        }
                        else
                        {
                                dezkomma(hilfsstring, (*Storage)[interval-1][mat-10]);
                                fprintf(outfile, "%s \t", hilfsstring);
                        }
                }

                fprintf(outfile, "\n");
        }
        fprintf(outfile, "\n");

        /* output of Unat */
        fprintf(outfile, "%s  \t", "consumption of Unat");
        for(interval=0; interval<=IT; interval++)
        {
                if(interval==0)
                        fprintf(outfile, "\t");
                else
                {
                        dezkomma(hilfsstring, (*Unat)[interval-1]);
                        fprintf(outfile, "%s \t", hilfsstring);
                }
        }
        fprintf(outfile, "\n");

        /* output of Unatt */
        fprintf(outfile, "%s  \t", "consumption of Unatt");
        for(interval=0; interval<=IT; interval++)
        {
                if(interval==0)
                        fprintf(outfile, "\t");
                else
                {
                        dezkomma(hilfsstring, (*Unatt)[interval-1]);
                        fprintf(outfile, "%s \t", hilfsstring);
                }
```

```
}
fprintf(outfile, "\n");

/* output of Tht */
fprintf(outfile, "%s  \t", "consumption of Tht");
for(interval=0; interval<=IT; interval++)
{
	if(interval==0)
		fprintf(outfile, "\t");
	else
		{
			dezkomma(hilfsstring, (*Tht)[interval-1]);
			fprintf(outfile, "%s \t", hilfsstring);
		}
}
fprintf(outfile, "\n");

/* output of Refabrication */
fprintf(outfile, "%s  \t", "Refabrication");
for(interval=0; interval<=IT; interval++)
{
	if(interval==0)
		fprintf(outfile, "\t");
	else
		{
			dezkomma(hilfsstring, (*Refab)[interval-1]);
			fprintf(outfile, "%s \t", hilfsstring);
		}
}
fprintf(outfile, "\n\n");

/* UirrWaste output */
fprintf(outfile, "%s  \t", "UirrWaste");
for(interval=0; interval<=IT; interval++)
{
	if(interval==0)
		{
			dezkomma(hilfsstring, UirrWasteResid);
			fprintf(outfile, "%s \t", hilfsstring);
		}
	else
		{
			dezkomma(hilfsstring, (*UirrWaste)[interval-1]);
			fprintf(outfile, "%s \t", hilfsstring);
		}
}
fprintf(outfile, "\n");

/* PuWaste output */
fprintf(outfile, "%s  \t", "PuWaste");
for(interval=0; interval<=IT; interval++)
{
	if(interval==0)
		{
			dezkomma(hilfsstring, PuWasteResid);
			fprintf(outfile, "%s \t", hilfsstring);
		}
	else
		{
			dezkomma(hilfsstring, (*PuWaste)[interval-1]);
			fprintf(outfile, "%s \t", hilfsstring);
		}
}
fprintf(outfile, "\n");

/* MaWaste output */
fprintf(outfile, "%s  \t", "MaWaste");
for(interval=0; interval<=IT; interval++)
{
	if(interval==0)
		{
			dezkomma(hilfsstring, MaWasteResid);
			fprintf(outfile, "%s \t", hilfsstring);
		}
	else
		{
			dezkomma(hilfsstring, (*MaWaste)[interval-1]);
			fprintf(outfile, "%s \t", hilfsstring);
```

```
                }
        }
        fprintf(outfile, "\n\n");

        /* Waste output */
        fprintf(outfile, "%s  \t", "Waste");
        for(interval=0; interval<=IT; interval++)
        {
                if(interval==0)
                        fprintf(outfile, "\t");
                else
                        {
                                dezkomma(hilfsstring, (*Waste)[interval-1]);
                                fprintf(outfile, "%s \t", hilfsstring);
                        }
        }
        fprintf(outfile, "\n");


}
```

/* filename: vleem2_error_output.c */

```
#include "vleem2_header.h"
#include "vleem2_capacity_header.h"
#include "vleem2_mass_header.h"

void error_output(void)
{
	FILE* outfile;
	char* class_names[3] = {"Resid", "Str", "Pw"};

	if(error == 0)
	/* no error */
		return;

	assert(outfile = fopen("error_output.txt", "w"));

	switch(error)
	{
	case ARGUMENT_ERROR:
		fprintf(outfile, "\nArgument error: too many arguments in command line!!\n");
		break;
	case OPEN_ERROR:
		fprintf(outfile, "File could not be opened!\nPlease check, if all output files
are closed!\n");
		break;
	case INPUT_TH_ERROR:
		fprintf(outfile, "Error in input of time horizon! Must be integer!\n");
		break;
	case INPUT_TYPENO_ERROR:
		fprintf(outfile, "Error in input of number of types! Must be integer!\n");
		break;
	case INPUT_FORMAT_ERROR:
		fprintf(outfile, "Error in input! Wrong format of space text!\n");
		break;
	case INPUT_NAME_ERROR:
		fprintf(outfile, "Error in input of typenames!\n");
		break;
	case INPUT_LT_ERROR:
		fprintf(outfile, "Error in input of lifetimes! Must be integer!\n");
		break;
	case INPUT_FLH_ERROR:
		fprintf(outfile, "Error in input of full load hours! Must be integer!\n");
		break;
	case INPUT_CT_ERROR:
		fprintf(outfile, "Error in input of cooling times! Must be integer!\n");
		break;
	case INPUT_LT_FLH_ERROR:
		fprintf(outfile, "Error in input! Lifetime must be a multiple of JT!\nFull load
hour must be smaller than 8760!\n");
		break;
	case INPUT_TTC_ERROR:
		fprintf(outfile, "Error in input of total capacity for one type at the
beginning!\n");
		break;
	case INPUT_TCC_ERROR:
		fprintf(outfile, "Error in input of total capacity for one class in one
period!\n");
		break;
	case INPUT_NEWCAP_ERROR:
		fprintf(outfile, "Error in input of new capacity for one type in one
period!\n");
		break;
	case INPUT_NEG_RESID_ERROR:
		fprintf(outfile, "Error in input of new capacity for one type of class
Resid!\n"
					"At least one type has negative capacity in
interval %d!\n", error_int);
		break;
	case INPUT_POSDIFF_ERROR:
		fprintf(outfile, "Error in input of new capacity for class %s in interval
%d!\n"
					"Please reduce your input about %6.1f GW!\n",
class_names[error_class], error_int, difference);
		break;
	case INPUT_NEGDIFF_ERROR:
		fprintf(outfile, "Error in input of new capacity for class %s in interval
%d!\n"
```

```
                                                "Please increase your input about %6.1f GW!\n",
class_names[error_class], error_int, (-1*difference));
                break;
        case INPUT_MATRIX_ERROR:
                fprintf(outfile, "Error in input of data matrix!\n");
                break;
        case INPUT_STORAGE_ERROR:
                fprintf(outfile, "Error in input of resid Storage!\n");
                break;
        case INPUT_FACTOR_ERROR:
                fprintf(outfile, "Error in input of loosing factors!\n");
                break;
        case INPUT_WASTE_ERROR:
                fprintf(outfile, "Error in input of loosing factors!\n");
                break;
        case INPUT_ANSWER_ERROR:
                fprintf(outfile, "Wrong answer!\n");
                break;

        default:
                ;
        }

        fclose(outfile);
}



/* results of part I and II for input of part III */
void mass_results(int ttypes, reactor_class** classes, float**** TMatin1, float**** TMatin2,
float**** TMatout1, float**** TMatout2, float** StorageResid,
                                float UirrWasteResid, float PuWasteResid, float MaWasteResid,
float VPU, float VUIRR, float VMA)
{
        int j, k;
        int interval;
        int type;
        substance mat;
        FILE* results;

        assert(results = fopen("vleem2_use_input.txt", "w"));

        fprintf(results, "/* time horizon (JI, JF, JT, IT): */\n%d\n%d\n%d\n%d\n\n", JI, JF,
JT, IT);

        fprintf(results, "/* total number of types: */\n%d\n\n", ttypes);

        fprintf(results, "/* reactor_types (name, lifetime, full_load_hour, cooling_time):
*/\n");
        for(j=RESID; j<=PW; j++)
        {
                for(k=0; k<(*classes)[j].number_of_types; k++)
                {
                        fprintf(results, "%s\n%d\n%.2f\n%d\n\n",
                                        (*classes)[j].types[k].name,
(*classes)[j].types[k].lifetime,
                                        (*classes)[j].types[k].full_load_hour,
(*classes)[j].types[k].cooling_time);
                }
        }

        fprintf(results, "TMatin1:\n");
        for(interval=0; interval<IT; interval++)
        {
                fprintf(results, "%d. interval:\n", interval+1);
                for(mat=UnatIn; mat<=HmIn; mat++)
                {
                        for(type=0; type<ttypes; type++)
                        {
                                fprintf(results, "%.2f\t", (*TMatin1)[interval][mat-1][type]);
                        }

                        fprintf(results, "\n");
                }
                fprintf(results, "\n");
        }

        fprintf(results, "TMatin2:\n");
```

```
	for(interval=0; interval<IT; interval++)
	{
		fprintf(results, "%d. interval:\n", interval+1);
		for(mat=UnatIn; mat<=HmIn; mat++)
		{
			for(type=0; type<ttypes; type++)
			{
				fprintf(results, "%.2f\t", (*TMatin2)[interval][mat-1][type]);
			}

			fprintf(results, "\n");
		}
		fprintf(results, "\n");
	}

	fprintf(results, "TMatout1:\n");
	for(interval=0; interval<IT; interval++)
	{
		fprintf(results, "%d. interval:\n", interval+1);
		for(mat=ThUOut; mat<=PurecOut; mat++)
		{
			for(type=0; type<ttypes; type++)
			{
				fprintf(results, "%.2f\t", (*TMatout1)[interval][mat-10][type]);
			}

			fprintf(results, "\n");
		}
		fprintf(results, "\n");
	}

	fprintf(results, "TMatout2:\n");
	for(interval=0; interval<IT; interval++)
	{
		fprintf(results, "%d. interval:\n", interval+1);
		for(mat=ThUOut; mat<=PurecOut; mat++)
		{
			for(type=0; type<ttypes; type++)
			{
				fprintf(results, "%.2f\t", (*TMatout2)[interval][mat-10][type]);
			}

			fprintf(results, "\n");
		}
		fprintf(results, "\n");
	}

	fprintf(results, "\nstorage (resid for mat ThUOut to PurecOut):\n");
	for(mat=ThUOut; mat<=PurecOut; mat++)
		fprintf(results, "%.2f\t", (*StorageResid)[mat-10]);

	fprintf(results, "\n\nwaste (resid for UirrWaste, PuWaste, MaWaste):\n%f\n%f\n%f\n\n",
			UirrWasteResid, PuWasteResid, MaWasteResid);

	fprintf(results, "losing factors (VPU, VUIRR, VMA):\n%.f\n%f\n%f\n", VPU, VUIRR, VMA);

	fclose(results);


}
```

5.2 VLEEM2_USE :

/* filename: vleem2_use_header.h */

```
#ifndef VLEEM2_USE_HEADER
#define VLEEM2_USE_HEADER


/* include-files */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <ctype.h>


/*  enumeration of the different types in a class of reactors  */
typedef int type_number;


/*  new type for a reactor-type  */
typedef struct{
       char* name;
       type_number nr;
       int lifetime;
       float full_load_hour;
       int cooling_time;
} reactor_type;


/*  enumeration of the substances  */
typedef enum{UnatIn = 1, ThIn, FpIn, UdeplIn, UirrIn,
                      MaIn, PufIn, PurecIn, HmIn, ThUOut,
                      FpOut, UdeplOut, UirrOut, MaOut, PufOut,
                      PurecOut} substance;


/* errors */
#define ARGUMENT_ERROR 1
#define INPUT_TIME_ERROR 2
#define INPUT_CAPF_ERROR 3

/* new time horizon */
int IN;                                                    /* number of short periods
per JT */
int N;                                                     /* total number of short
periods */
float BT;                                                  /* length of a short period
*/

/* error */
int error;

/* input function */
void input_use(FILE* infile_use, FILE* infile_capf, int* JI, int* JF, int* JT, int* IT,
                        int* ttypes, reactor_type** r_types, float**** TMatin1, float****
TMatin2,
                        float**** TMatout1, float**** TMatout2, float** Storage, float*
UirrWaste,
                        float* PuWaste, float* MaWaste, float* VPU, float* VUIRR, float* VMA,
float*** Capf);

/* memory allocation */
void memory_allocation(int ttypes, float*** DStorage, float*** DMatin, float** DUnat,
                                     float** DUnatt, float** DTht, float** Refabr, float**
DUirrWaste,
                                     float** DPuWaste, float** DMaWaste, float** DWaste);

/* calculation of mass for short periods */
void mass_calculation(int ttypes, int IT, reactor_type** r_types, float**** TMatin1, float****
TMatin2,
                                   float**** TMatout1, float**** TMatout2, float**
Storage, float*** Capf,
                                   float*** DStorage, float*** DMatin, float** DUnat,
float** DUnatt, float** DTht,
                                   float** Refabr);
```

```
/* calculation of waste for short periods */
void waste_calculation(float* UirrWaste, float* PuWaste, float* MaWaste, float VPU, float
VUIRR,
                                        float VMA, float** DUirrWaste, float** DPuWaste,
float** DMaWaste,
                                        float** DWaste, float*** DMatin, float*** DStorage);

/* output function */
void use_output(int JI, int ttypes, float*** Capf, float*** DStorage, float** Storage,
float*** DMatin, float** DUnat,
                      float** DUnatt, float** DTht, float** Refabr, float** DUirrWaste, float
UirrWaste,
                      float** DPuWaste, float PuWaste, float** DMaWaste, float MaWaste,
float** DWaste);

/* function for error output */
void error_output(void);


#endif
```

/* filename: vleem2_use.c */

```
#include "vleem2_use_header.h"

/* new time horizon */
int IN;                                                    /* number of short periods
per JT */
int N;                                                     /* total number of short
periods */
float BT;                                                  /* length of a short period
*/

/* error */
int error;

int main(int argc, char** argv)
{

        /* data from vleem2_mass */
        /* time horizon */
        int JI, JF, JT, IT;

        /* total number of different types of reactor */
        int ttypes;

        /* types of reactor */
        reactor_type* r_types;

        /*  incoming substance for each type and each material per period  */
        float*** TMatin1;
        float*** TMatin2;

        /*  outcoming substance for each type and each material per period  */
        float*** TMatout1;
        float*** TMatout2;

        /*  Storage for ThUOut to PrecOut in first period */
        float* Storage;

        /*  Uirr waste in first period */
        float UirrWaste;

        /*  Pu waste in first period */
        float PuWaste;

        /*  Ma waste in first period */
        float MaWaste;

        /*  loosing factors  */
        float VPU, VUIRR, VMA;

        /* data for new calculation */
        /* capacity factor for each type and each short period */
        float** Capf;

        /* storage for ThUOut to PurecOut for each short period */
        float** DStorage;

        /* Matin for short periods */
        float** DMatin;

        /* Unat for short periods */
        float* DUnat;

        /* Unatt for short periods */
        float* DUnatt;

        /* Tht for short periods */
        float* DTht;

        /* Refabr for short periods */
        float* Refabr;

        /* UirrWaste for short periods */
        float* DUirrWaste;

        /* PuWaste for short periods */
        float* DPuWaste;
```

```
	/* MaWaste for short periods */
	float* DMaWaste;

	/* Waste for short periods */
	float* DWaste;


	FILE* infile_use;
	FILE* infile_capf;

	if(argc!=3)
		error = ARGUMENT_ERROR;
	else
	{
		assert(infile_use = fopen(argv[1], "r"));
		assert(infile_capf = fopen(argv[2], "r"));
	}


	if(error == 0)
	{
		input_use(infile_use, infile_capf, &JI, &JF, &JT, &IT, &ttypes, &r_types,
&TMatin1, &TMatin2,
				&TMatout1, &TMatout2, &Storage, &UirrWaste, &PuWaste,
&MaWaste, &VPU, &VUIRR, &VMA, &Capf);

		if(error == 0)
		{
			memory_allocation(ttypes, &DStorage, &DMatin, &DUnat, &DUnatt, &DTht,
&Refabr,
							&DUirrWaste, &DPuWaste, &DMaWaste,
&DWaste);

			mass_calculation(ttypes, IT, &r_types, &TMatin1, &TMatin2, &TMatout1,
&TMatout2, &Storage, &Capf, &DStorage,
							&DMatin, &DUnat, &DUnatt, &DTht,
&Refabr);

			waste_calculation(&UirrWaste, &PuWaste, &MaWaste, VPU, VUIRR,
							VMA, &DUirrWaste, &DPuWaste, &DMaWaste,
&DWaste, &DMatin, &DStorage);

			use_output(JI, ttypes, &Capf, &DStorage, &Storage, &DMatin, &DUnat,
&DUnatt, &DTht, &Refabr,
					&DUirrWaste, UirrWaste, &DPuWaste, PuWaste, &DMaWaste,
MaWaste, &DWaste);
		}

		fclose(infile_use);
		fclose(infile_capf);

	}

	if(error != 0)
		error_output();


}
```

/* filename: vleem2_use_memory_allocation.c */

```
#include "vleem2_use_header.h"

void memory_allocation(int ttypes, float*** DStorage, float*** DMatin, float** DUnat, float**
DUnatt, float** DTht, float** Refabr,
                                        float** DUirrWaste, float** DPuWaste, float**
DMaWaste, float** DWaste)

{
        int i;

        /* DStorage */
        assert((*DStorage) = (float**) malloc(N * sizeof(float)));
        for(i=0; i<N; i++)
                assert((*DStorage)[i] = (float*) malloc(7 * sizeof(float)));


        /* DMatin */
        assert((*DMatin) = (float**) malloc(N * sizeof(float)));
        for(i=0; i<N; i++)
                assert((*DMatin)[i] = (float*) malloc(9 * sizeof(float)));

        /* DUnat */
        assert((*DUnat) = (float*) malloc(N * sizeof(float)));

        /* DUnatt */
        assert((*DUnatt) = (float*) malloc(N * sizeof(float)));

        /* DTht */
        assert((*DTht) = (float*) malloc(N * sizeof(float)));

        /* Refabr */
        assert((*Refabr) = (float*) malloc(N * sizeof(float)));

        /* DUirrWaste */
        assert((*DUirrWaste) = (float*) malloc(N * sizeof(float)));

        /* DPuWaste */
        assert((*DPuWaste) = (float*) malloc(N * sizeof(float)));

        /* DMaWaste */
        assert((*DMaWaste) = (float*) malloc(N * sizeof(float)));

        /* DWaste */
        assert((*DWaste) = (float*) malloc(N * sizeof(float)));
}
```

/* filename: vleem2_use_input */

```
#include "vleem2_use_header.h"

void input_use(FILE* infile_use, FILE* infile_capf, int* JI, int* JF, int* JT, int* IT, int*
ttypes,
                        reactor_type** r_types, float**** TMatin1, float**** TMatin2,
float**** TMatout1,
                        float**** TMatout2,float** Storage, float* UirrWaste, float* PuWaste,
float* MaWaste,
                        float* VPU, float* VUIRR, float* VMA, float*** Capf)
{

        int i,j,k;
        char helpline[4000];
        int character;

        /**********  input of vleem2_mass output  **********/

        /* scanf of comment-line in input */
        assert(fgets(helpline, 400, infile_use)!=NULL);

        /* time horizon */
        assert(fscanf(infile_use, "%d%d%d%d", JI, JF, JT, IT) == 4);

        /* space */
        while(isspace(character = fgetc(infile_use)));
        ungetc(character, infile_use);

        /* scanf of comment-line in input */
        assert(fgets(helpline, 400, infile_use)!=NULL);

        /* total number of types */
        assert(fscanf(infile_use, "%d", ttypes) == 1);

        /* space */
        while(isspace(character = fgetc(infile_use)));
        ungetc(character, infile_use);

        /* scanf of comment-line in input */
        assert(fgets(helpline, 400, infile_use)!=NULL);

        /* reactor_types */
        assert((*r_types) = (reactor_type*) malloc((*ttypes) * sizeof(reactor_type)));
        for(i=0; i<(*ttypes); i++)
        {
                (*r_types)[i].nr = i+1;
                assert((*r_types)[i].name = (char*) malloc(20*sizeof(char)));
                assert(fscanf(infile_use, "%s%d%f%d", (*r_types)[i].name,
&((*r_types)[i].lifetime),
                        &((*r_types)[i].full_load_hour), &((*r_types)[i].cooling_time)) ==
4);
        }

        /* space */
        while(isspace(character = fgetc(infile_use)));
        ungetc(character, infile_use);

        /* scanf of comment-line in input */
        assert(fgets(helpline, 400, infile_use)!=NULL);

        /* scanf of comment-line in input */
        assert(fgets(helpline, 400, infile_use)!=NULL);

        /* TMatin1 */
        assert((*TMatin1) = (float***) malloc((*IT)*sizeof(float**)));
        for(i=0; i<(*IT); i++)
        {
                assert((*TMatin1)[i] = (float**) malloc(9*sizeof(float*)));

                for(j=0; j<9; j++)
                {
                        assert((*TMatin1)[i][j] = (float*) malloc((*ttypes)*sizeof(float)));
                        for(k=0; k<(*ttypes); k++)
                                assert(fscanf(infile_use, "%f", &((*TMatin1)[i][j][k])) == 1);
                }
                /* space */
                while(isspace(character = fgetc(infile_use)));
```

```
	ungetc(character, infile_use);

	/* scanf of comment-line in input */
	assert(fgets(helpline, 400, infile_use)!=NULL);
}

/* scanf of comment-line in input */
assert(fgets(helpline, 400, infile_use)!=NULL);

/* TMatin2 */
assert((*TMatin2) = (float***) malloc((*IT)*sizeof(float**)));
for(i=0; i<(*IT); i++)
{
	assert((*TMatin2)[i] = (float**) malloc(9*sizeof(float*)));

	for(j=0; j<9; j++)
	{
		assert((*TMatin2)[i][j] = (float*) malloc((*ttypes)*sizeof(float)));
		for(k=0; k<(*ttypes); k++)
			assert(fscanf(infile_use, "%f", &((*TMatin2)[i][j][k])) == 1);
	}
	/* space */
	while(isspace(character = fgetc(infile_use)));
	ungetc(character, infile_use);

	/* scanf of comment-line in input */
	assert(fgets(helpline, 400, infile_use)!=NULL);
}

/* scanf of comment-line in input */
assert(fgets(helpline, 400, infile_use)!=NULL);

/* TMatout1 */
assert((*TMatout1) = (float***) malloc((*IT)*sizeof(float**)));
for(i=0; i<(*IT); i++)
{
	assert((*TMatout1)[i] = (float**) malloc(7*sizeof(float*)));

	for(j=0; j<7; j++)
	{
		assert((*TMatout1)[i][j] = (float*) malloc((*ttypes)*sizeof(float)));
		for(k=0; k<(*ttypes); k++)
			assert(fscanf(infile_use, "%f", &((*TMatout1)[i][j][k])) == 1);
	}
	/* space */
	while(isspace(character = fgetc(infile_use)));
	ungetc(character, infile_use);

	/* scanf of comment-line in input */
	assert(fgets(helpline, 400, infile_use)!=NULL);
}

/* scanf of comment-line in input */
assert(fgets(helpline, 400, infile_use)!=NULL);

/* TMatout2 */
assert((*TMatout2) = (float***) malloc((*IT)*sizeof(float**)));
for(i=0; i<(*IT); i++)
{
	assert((*TMatout2)[i] = (float**) malloc(7*sizeof(float*)));

	for(j=0; j<7; j++)
	{
		assert((*TMatout2)[i][j] = (float*) malloc((*ttypes)*sizeof(float)));
		for(k=0; k<(*ttypes); k++)
			assert(fscanf(infile_use, "%f", &((*TMatout2)[i][j][k])) == 1);
	}
	/* space */
	while(isspace(character = fgetc(infile_use)));
	ungetc(character, infile_use);

	/* scanf of comment-line in input */
	assert(fgets(helpline, 400, infile_use)!=NULL);
}

/* Storage */
assert((*Storage) = (float*) malloc(7*sizeof(float)));
for(i=0; i<7; i++)
```

```
		assert(fscanf(infile_use,"%f", &((*Storage)[i])) == 1);

	/* space */
	while(isspace(character = fgetc(infile_use)));
	ungetc(character, infile_use);

	/* scanf of comment-line in input */
	assert(fgets(helpline, 400, infile_use)!=NULL);

	/* waste */
	assert(fscanf(infile_use,"%f%f%f", UirrWaste, PuWaste, MaWaste) == 3);

	/* space */
	while(isspace(character = fgetc(infile_use)));
	ungetc(character, infile_use);

	/* scanf of comment-line in input */
	assert(fgets(helpline, 400, infile_use)!=NULL);

	/* loosing factors */
	assert(fscanf(infile_use,"%f%f%f", VPU, VUIRR, VMA) == 3);



	/**********  input of new time horizon  **********/
	printf("Number of short periods for JT:\n");
	if(scanf("%d", &IN) != 1)
		error = INPUT_TIME_ERROR;
	else
	{
		N = (*IT) * IN;
		BT = (float)(*JT) / (float)IN;
	}


	/**********  input of capf  **********/

	/* memory allocation and input of capf */
	if(error == 0)
	{
		assert((*Capf) = (float**) malloc(N * sizeof(float*)));
		for(i=0; i<N; i++)
			assert((*Capf)[i] = (float*) malloc((*ttypes) * sizeof(float)));

		/* scanf of comment-line in input */
		assert(fgets(helpline, 4000, infile_capf)!=NULL);
		for(j=0; j<(*ttypes); j++)
		{

			if(error)
				break;

			if(fscanf(infile_capf,"%[^0123456789]", helpline) != 1)
				error = INPUT_CAPF_ERROR;

			for(i=0; i<N; i++)
			{
				if(fscanf(infile_capf, "%f", &((*Capf)[i][j])) != 1)
					error = INPUT_CAPF_ERROR;
			}

			/* space */
			while(isspace(character = fgetc(infile_capf)));
			ungetc(character, infile_capf);
		}
	}
}
```

/* filename: vleem2_use_calculation */

```
#include "vleem2_use_header.h"

void mass_calculation(int ttypes, int IT, reactor_type** r_types, float**** TMatin1, float****
TMatin2,
                                        float**** TMatout1, float**** TMatout2, float**
Storage, float*** Capf,
                                        float*** DStorage, float*** DMatin, float** DUnat,
float** DUnatt,
                                        float** DTht, float** Refabr)
{

        int sint;                                               /* short interval */
        int lint;                                               /* long interval */
        int type;
        substance mat;
        float factor;

        /* initialisation of DStorage (Resids) */
        for(mat=ThUOut; mat<=PurecOut; mat++)
        {
                (*DStorage)[0][mat-10] = (*Storage)[mat-10];
        }


        /* new data for TMatin and TMatout */
        for(lint=1; lint<=IT; lint++)
        {
                for(type=1; type<=ttypes; type++)
                {
                        for(mat=UnatIn; mat<=PurecOut; mat++)
                        {
                                if(mat<ThUOut)
                                {
                                        (*TMatin1)[lint-1][mat-1][type-1] /= IN;
                                        (*TMatin2)[lint-1][mat-1][type-1] /= IN;
                                }
                                else
                                {
                                        (*TMatout1)[lint-1][mat-10][type-1] /= IN;
                                        (*TMatout2)[lint-1][mat-10][type-1] /= IN;
                                }
                        }
                }
        }

        (*DUnatt)[0] = 0;
        (*DTht)[0] = 0;
        for(sint=1; sint<=N; sint++)
        {
                for(type=1; type<=ttypes; type++)
                {
                        (*Capf)[sint-1][type-1] *= (float)8.76/((*r_types)[type-
1].full_load_hour);
                }

                lint = 1+ (sint-1)/IN;

                for(mat=UnatIn; mat<=HmIn; mat++)
                {
                        (*DMatin)[sint-1][mat-1] = 0;

                        for(type=1; type<=ttypes; type++)
                        {
                                (*DMatin)[sint-1][mat-1] += (*TMatin1)[lint-1][mat-1][type-1] *
(*Capf)[sint-1][type-1]
                                                                                        +
(*TMatin2)[lint-1][mat-1][type-1];
                        }
                }

                if(sint == 1)
                {
                        (*DUnatt)[sint-1] += (*DMatin)[sint-1][UnatIn-1];
                        (*DTht)[sint-1] += (*DMatin)[sint-1][ThIn-1];
                }
                else
```

```
		{
			(*DUnatt)[sint-1] = (*DUnatt)[sint-2] + (*DMatin)[sint-1][UnatIn-1];
			(*DTht)[sint-1] = (*DTht)[sint-2] + (*DMatin)[sint-1][ThIn-1];
		}

		(*Refabr)[sint-1] = ((*DMatin)[sint-1][HmIn-1]) / BT;

		(*DUnat)[sint-1] = (*DMatin)[sint-1][UnatIn-1];


		for(mat=ThUOut; mat<=PufOut; mat++)
		{
			factor = 0;
			for(type=1; type<=ttypes; type++)
			{
				if((lint-(*r_types)[type-1].cooling_time) > 0)
					factor += (*TMatout1)[lint-(*r_types)[type-
1].cooling_time -1][mat-10][type-1] * (*Capf)[sint-1][type-1]
							+ (*TMatout2)[lint-(*r_types)[type-
1].cooling_time -1][mat-10][type-1];
			}

			if(sint==1)
				(*DStorage)[sint-1][mat-10] += factor;
			else
				(*DStorage)[sint-1][mat-10] = (*DStorage)[sint-2][mat-10] +
factor;

			if(mat>ThUOut)
				(*DStorage)[sint-1][mat-10] -= (*DMatin)[sint-1][mat-9];
		}

		/* mat==PurecOut */
		if(sint==1)
			(*DStorage)[sint-1][mat-10] += factor - (*DMatin)[sint-1][PufIn-1];
		else
			(*DStorage)[sint-1][mat-10] = (*DStorage)[sint-2][mat-10] + factor -
(*DMatin)[sint-1][PufIn-1];

		factor = 0;
		for(type=1; type<=ttypes; type++)
		{
			if((lint-(*r_types)[type-1].cooling_time) > 0)
				factor += (*TMatout1)[lint-(*r_types)[type-1].cooling_time -
1][mat-10][type-1] * (*Capf)[sint-1][type-1]
					+ (*TMatout2)[lint-(*r_types)[type-
1].cooling_time -1][mat-10][type-1];
		}

		(*DStorage)[sint-1][mat-10] += factor - (*DMatin)[sint-1][mat-9];
	}
}



void waste_calculation(float* UirrWaste, float* PuWaste, float* MaWaste, float VPU, float
VUIRR,
					float VMA, float** DUirrWaste, float** DPuWaste,
float** DMaWaste,
					float** DWaste, float*** DMatin, float*** DStorage)
{

	int sint;

	/* Resids */
	(*DUirrWaste)[0] = *UirrWaste;
	(*DPuWaste)[0] = *PuWaste;
	(*DMaWaste)[0] = *MaWaste;

	for(sint=1; sint<=N; sint++)
	{
		if(sint==1)
		{
			(*DUirrWaste)[0] += VUIRR * (*DMatin)[sint-1][UirrIn-1];
			(*DPuWaste)[0] += VPU * ((*DMatin)[sint-1][PufIn-1] + (*DMatin)[sint-
1][PurecIn-1]);
			(*DMaWaste)[0] += VMA * (*DMatin)[sint-1][MaIn-1];
		}
```

```
                else
                {
                        (*DUirrWaste)[sint-1] = (*DUirrWaste)[sint-2] + VUIRR * (*DMatin)[sint-
1][UirrIn-1];
                        (*DPuWaste)[sint-1] = (*DPuWaste)[sint-2] + VPU * ((*DMatin)[sint-
1][PufIn-1] + (*DMatin)[sint-1][PurecIn-1]);
                        (*DMaWaste)[sint-1] = (*DMaWaste)[sint-2] + VMA * (*DMatin)[sint-
1][MaIn-1];
                }

                (*DWaste)[sint-1] = (*DStorage)[sint-1][ThUOut-10] + (*DStorage)[sint-1][FpOut-
10] + (*DUirrWaste)[sint-1] + (*DPuWaste)[sint-1] + (*DMaWaste)[sint-1];
        }

}
```

/* filename: vleem2_use_output.c */

```
#include "vleem2_use_header.h"


void dezkomma(char* str, float wert)
{
	int index=0;

	sprintf(str, "%.1f", wert);
	while(str[index] != '\0')
	{
		if(str[index] == '.')
		{
			str[index] = ',';
			break;
		}
		index++;
	}

}



void use_output(int JI, int ttypes, float*** Capf, float*** DStorage, float** Storage,
float*** DMatin, float** DUnat,
			float** DUnatt, float** DTht, float** Refabr, float** DUirrWaste, float
UirrWaste,
			float** DPuWaste, float PuWaste, float** DMaWaste, float MaWaste,
float** DWaste)
{
	int interval;
	substance mat;
	char *matname[] = { "UnatIn","ThIn","FpIn","UdeplIn","UirrIn","MaIn","PufIn","PurecIn",

	"HmIn","ThUOut","FpOut","UdeplOut","UirrOut","MaOut","PufOut","PurecOut"};
	FILE* outfile;
	char hilfsstring[20];

	assert(outfile = fopen("output_use.xls", "w"));

	/* column headlines */
	fprintf(outfile, "%s  \t", "Interval");
	for(interval=0; interval<=N; interval++)
	{
		if(interval==0)
			fprintf(outfile, "%s/%d\t", "RESID", JI);
		else
			fprintf(outfile, "%d \t", interval);
	}
	fprintf(outfile, "\n\n");

	/* output of DMatin */
	for(mat=UnatIn; mat<=HmIn; mat++)
	{
		fprintf(outfile, "%s  \t", matname[mat-1]);
		for(interval=0; interval<=N; interval++)
			{
				if(interval==0)
					fprintf(outfile, "\t");
				else
				{
					dezkomma(hilfsstring, (*DMatin)[interval-1][mat-1]);
					fprintf(outfile, "%s \t", hilfsstring);
				}
			}
		fprintf(outfile, "\n\n");
	}

	/* output of DStorage */
	for(mat=ThUOut; mat<=PurecOut; mat++)
	{
		fprintf(outfile, "%s dstorage \t", matname[mat-1]);
		for(interval=0; interval<=N; interval++)
		{
			if(interval==0)
				{
					dezkomma(hilfsstring, (*Storage)[mat-10]);
```

```
                        fprintf(outfile, "%s \t", hilfsstring);
                }
        else
                {
                        dezkomma(hilfsstring, (*DStorage)[interval-1][mat-10]);
                        fprintf(outfile, "%s \t", hilfsstring);
                }
        }

        fprintf(outfile, "\n");
}
fprintf(outfile, "\n");

/* output of DUnat */
fprintf(outfile, "%s  \t", "consumption of DUnat");
for(interval=0; interval<=N; interval++)
{
        if(interval==0)
                fprintf(outfile, "\t");
        else
                {
                        dezkomma(hilfsstring, (*DUnat)[interval-1]);
                        fprintf(outfile, "%s \t", hilfsstring);
                }
}
fprintf(outfile, "\n");

/* output of DUnatt */
fprintf(outfile, "%s  \t", "consumption of DUnatt");
for(interval=0; interval<=N; interval++)
{
        if(interval==0)
                fprintf(outfile, "\t");
        else
                {
                        dezkomma(hilfsstring, (*DUnatt)[interval-1]);
                        fprintf(outfile, "%s \t", hilfsstring);
                }
}
fprintf(outfile, "\n");

/* output of DTht */
fprintf(outfile, "%s  \t", "consumption of DTht");
for(interval=0; interval<=N; interval++)
{
        if(interval==0)
                fprintf(outfile, "\t");
        else
                {
                        dezkomma(hilfsstring, (*DTht)[interval-1]);
                        fprintf(outfile, "%s \t", hilfsstring);
                }
}
fprintf(outfile, "\n");

/* output of Refabrication */
fprintf(outfile, "%s  \t", "Refabrication");
for(interval=0; interval<=N; interval++)
{
        if(interval==0)
                fprintf(outfile, "\t");
        else
                {
                        dezkomma(hilfsstring, (*Refabr)[interval-1]);
                        fprintf(outfile, "%s \t", hilfsstring);
                }
}
fprintf(outfile, "\n\n");

/* output of DUirrWaste */
fprintf(outfile, "%s  \t", "DUirrWaste");
for(interval=0; interval<=N; interval++)
{
        if(interval==0)
                {
                        dezkomma(hilfsstring, UirrWaste);
                        fprintf(outfile, "%s \t", hilfsstring);
                }
```

```
                else
                        {
                                dezkomma(hilfsstring, (*DUirrWaste)[interval-1]);
                                fprintf(outfile, "%s \t", hilfsstring);
                        }
        }
        fprintf(outfile, "\n");

        /* output of DPuWaste */
        fprintf(outfile, "%s  \t", "DPuWaste");
        for(interval=0; interval<=N; interval++)
        {
                if(interval==0)
                        {
                                dezkomma(hilfsstring, PuWaste);
                                fprintf(outfile, "%s \t", hilfsstring);
                        }
                else
                        {
                                dezkomma(hilfsstring, (*DPuWaste)[interval-1]);
                                fprintf(outfile, "%s \t", hilfsstring);
                        }
        }
        fprintf(outfile, "\n");

        /* output of DMaWaste */
        fprintf(outfile, "%s  \t", "DMaWaste");
        for(interval=0; interval<=N; interval++)
        {
                if(interval==0)
                        {
                                dezkomma(hilfsstring, MaWaste);
                                fprintf(outfile, "%s \t", hilfsstring);
                        }
                else
                        {
                                dezkomma(hilfsstring, (*DMaWaste)[interval-1]);
                                fprintf(outfile, "%s \t", hilfsstring);
                        }
        }
        fprintf(outfile, "\n\n");

        /* output of DWaste */
        fprintf(outfile, "%s  \t", "DPuWaste");
        for(interval=0; interval<=N; interval++)
        {
                if(interval==0)
                        fprintf(outfile, "\t");
                else
                        {
                                dezkomma(hilfsstring, (*DWaste)[interval-1]);
                                fprintf(outfile, "%s \t", hilfsstring);
                        }
        }
        fprintf(outfile, "\n");


        fclose(outfile);
}


void error_output(void)
{
        FILE* outfile;

        if(error == 0)
        /* no error */
                return;

        assert(outfile = fopen("error_output_use.txt", "w"));

        switch(error)
        {
        case ARGUMENT_ERROR:
                fprintf(outfile, "\nArgument error: too many arguments in command line!!\n");
                break;
        case INPUT_TIME_ERROR:
                fprintf(outfile, "Input error: number of short periods must be integer!\n");
```

```
            break;
        case INPUT_CAPF_ERROR:
            fprintf(outfile, "Input error: wrong data for capf!\n");
            break;
        default: ;
        }

        fclose(outfile);
}
```

VLEEM-2 **ATTACHMENT** **FZJ/STE**

Operational Users´ Guide
for the
PC-based Nuclear Mass Flow Programmes
VLEEM2_MASS
(Capacity- and Mass Balance-Computations)
and
VLEEM2_USE
(Operational Mass Balance Computations)

Sylvia Gasper, Gerhard Kolb

November 2003

1. Introduction

VLEEM2_MASS: Allows dynamic capacity- and mass flow computations for a nuclear reactor park with "coarse" constant time steps (e.g., 10 years) within a deliberately large time horizon (e.g., 100 years).

VLEEM2_USE: Produces mass flow computations with deliberately small constant time steps, but with variable load factors, based on the input data and results of VLEEM2_MASS: It operates with the same reactor configurations.

The explanations below refer only to the **practical** execution of the two modules of the **Nuclear Mass Balance Model (NMBM)**, after its input has been prepared as briefly explained in the NMBM description in the **Annex.**

This input includes mainly:

- The time horizon (e.g., 2000-2100)
- Size of the "coarse" time step (e.g., 10 years, i.e. 10 time steps)
- 3 "classes" of reactors:
  1. RESID: Nuclear plants (NP) already existing at beginning of the time horizon
  2. EL: Electricity producers
  3. PH: Process heat producers

  All classes are expressed in electrical units: Gigawatt-electric – GW-el
  Total capacity = RESID + EL + PH (GW-el) in each time step
- Definition of the reactor types for each class
- For each RESID-reactor type the decreasing capacity for each time step (until the capacity is – and remains – zero).
- Time sequence of the total capacities for EL and PW
- Time sequence of new plant additions for each type in each class [Type(Class)]: CAPnew (Class, Type(Class); I) for each time step I: Must be conform to the time sequence of the total capacities for EL and PW ! (Is checked by the PC-programme !)

These data are basic definition items of the scenarios and are organised in an EXCEL-input file, here called "vleem2-input". An example of it is shown in Table 1, including explanations of the data contents.

Additionally the following reactor-specific data are required:

- Each reactor type has 3 data blocks:
  - First core (t/GW-el)
  - Rest core (t/GW-el)
  - "Reloads" (charges) and discharges (t/TW_{el}h)

  Each reactor has 16 components for these 3 items:
  - Columns 1 – 9 for charges,
  - Columns 10 – 16 for discharges.

  Matrix elements without a number are set to zero.

Table 2 shows a data matrix of reactor types (to be prepared as EXCEL-file "Matrix.xls") wit
h data (except zeros in empty spaces), without reference to classes, except the RESID-reactors (already existing at the beginning of the time horizon), marked with the post-fix "-R". In the case presented here the two types of LWR-R reactors have the same specific mass flow data as the LWR-reactors newly built within the time horizon. The reactors in Table 2 must be ordered in the same sequence as in Table 1.

Furthermore the following information items are needed:

- Initial values for storages (EXCEL-file "Storage.xls") (in tonnes heavy metal – t HM) of Th/U (discharges for final disposal), fission products (tonnes of fission products – t FP), depleted U (U-depl) from enrichment plants, irradiated U (U-irr – for potential reuse), minor actinides (MA), fresh (i.e. not or once recycled) Pu, and total Pu (fresh and multiple-recycled Pu). Storage.xls contains also loss-factors for reprocessing of Pu, irradiated U (U-irr) and MA. Target values for these latter items are less than 1 %, near to 0.001 (Table 3).
- EXCEL file "capf.xls": Capacity factors for computations with "small" time steps (integer fraction n of the "coarse" time steps) with

  $$0 \leq \text{capf} \leq 1$$

  for each reactor type (same sequence as in Table 1) and small time step.
  Table 4 shows such a matrix "capf.xls" for the case of 10 coarse and 20 small time steps (n = 2). The matrix must always correspond exactly to the total number of small time steps (columns = n x number of coarse time steps), not less and also not more! The value for each reactor and time step can be set to any value independent from the values taken for the "coarse" time steps where they are only dependent from the reactor type, but constant over time (and expressed in hours per year).

2. Requirements for the execution of the programme

Required input files:

- EXCEL-file "vleem2-input" of the basic scenario input data (Table 1)
- EXCEL-file "Matrix.xls" with the specific mass flow data for the reactors (Table 2)
- EXCEL-file "Storage.xls" with initial values for 7 storages and 3 loss factors for re-processing (Table 3)
- EXCEL-file "capf.xls" with capacity factors for computations with "small" time steps (Table 4)

These 4 EXCEL-files, if required, are to be changed in EXCEL, but must be saved prior the first execution and after each data change as ".txt"- or ".csv"-file:

Year of beginning:	2000									
Year of ending:	2100									
Length of interval (a):	10									
Number of types for class RESID:	2									
Number of types for class EL:	4									
Number of types for class PH:	2									
Types of class RESID:	**Name**	**Lifetime (a)**	**flh (h/a)**	**Cooling-time (intervals)**						
	LWR-UOX-R	40	7884	1						
	LWR-MOX-R	40	7884	1						
Types of class EL	**Name**	**Lifetime (a)**	**flh (h/a)**	**Cooling-time (intervals)**						
	LWR-UOX	40	7884	1						
	LWR-MOX	40	7884	1						
	CAPRA	40	7884	1						
	ADS	40	7884	1						
Types of class PH:	**Name**	**Lifetime (a)**	**flh (h/a)**	**Cooling-time (intervals)**						
	HTR-UOX	40	7884	1						
	HTR-MOX	40	7884	1						
Capacities at the beginning of the first interval (GW-el):	**LWR-UOX-R**	**LWR-MOX-R**	**LWR-UOX**	**LWR-MOX**	**CAPRA**	**ADS**	**HTR-UOX**	**HTR-MOX**		
	111	9	0	0	0	0	0	0		
Total capacities (GW-el) at the end of intervals	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
class RESID	80	40	0	0	0	0	0	0	0	0
class EL	40	80	120	160	200	240	280	320	360	400
class PH	0	40	80	120	160	200	240	280	320	360

Closure for RESID at the end / new capacities at the beginning of intervals for EL and PH (GW-el)		1	2	3	4	5	6	7	8	9	10
RESID	LWR-UOX-R	37	37	37	0	0	0	0	0	0	0
	LWR-MOX-R	3	3	3	0	0	0	0	0	0	0
EL	LWR-UOX	40	37	37	37	75	71	69	66	91	95
	LWR-MOX	0	3	3	3	5	5	5	9	16	10
	CAPRA	0	0	0	0	0	3	3	2	0	0
	ADS	0	0	0	0	0	1	3	3	13	15
PH	HTR-UOX	0	40	40	37	37	75	75	75	75	115
	HTR-MOX	0	0	0	3	3	5	5	5	5	5

Table 1: Basic definition items of the scenarios are organised in an EXCEL-input file, called "vleem2-input.xls". The presented example is also content of the other Tables 2-4 shown below.

Comments to Table 1:

- The input-file starts with some basic definitions (time horizon, length of a "coarse" interval, in years)
- Next are the numbers of reactor types in the classes RESID, EL and PH.
- Each reactor type in these 3 classes is defined by name, lifetime (years), full load hours per year, and out-of-pile time ("cooling time") in numbers of "coarse" intervals.
- The RESID-reactors (with the post-fix "-R") have (by definition) existing capacities (GW-el) at the beginning of the time horizon. The EL-and PH-types have zero initial capacities (by definition). The RESID-reactors can die out only, there are no later capacity additions for them.
- For each of the 3 classes the time sequence of their total capacities (GW-el) is defined.
- For each RESID-reactor the number of retired capacities (in GW-el) per time interval is given ("closure"). It must correspond to the predefined total capacity dynamics and is checked by the programme. An error message is produced identifying the time step of the error. After correcting it the programme has to be restarted (see Chapter 3).
- For each EL- and PH-reactor type the newly built capacity is given for each time step under consideration of the retired capacities (defined by the lifetimes). It must correspond to the predefined total capacity dynamics and is checked by the programme. An error message is produced identifying the time step of the error. After correcting it the programme has to be restarted (see Chapter 3).

Column number: 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	U-nat-in	Th-in	FP-in	U-depl-in	U-irr-in	MA-in	Pu-f-in	Pu-rec-in	HM-in	Th/U-out	FP-out	U-depl-out	U-irr-out	MA-out	Pu-f-out	Pu-rec-out
Reload t/TWh																
LWR-UOX-R	18.7										0.1131	16.4	2.18	0.0023	0.0236	
LWR-MOX-R				1.6411			0.2132		1.854		0.1168		1.5725	0.013		0.152
LWR-UOX	18.7										0.1131	16.4	2.18	0.0023	0.0236	
LWR-MOX				1.6411			0.2132		1.854		0.1168		1.5725	0.013		0.152
CAPRA				0.05	0.2595			0.2475	0.557		0.1225		0.2598	0.015		0.1797
ADS					0.0056	0.0786		0.3953	0.4795		0.1192		0.0056	0.0656		0.2811
HTR-UOX	19.98										0.1041	18.72	1.1207		0.00156	
HTR-MOX	14.27	0.6382					0.1061		0.8143	0.606	0.1073	14.2	0.0112	0.0108		0.0329
First core t/GW																
LWR-UOX-R	491.44											431				
LWR-MOX-R				43.13			5.6		48.73							
LWR-UOX	491.44											431				
LWR-MOX				43.13			5.6		48.73							
CAPRA				1.59	8.254			7.873	17.717							
ADS					0.124	1.742		8.763	10.629							
HTR-UOX	242.67											214.42				
HTR-MOX	724.55	38.41					3.92		7.52			720.95				
Rest core t/GW																
LWR-UOX-R											2.142		85.209	0.0436	0.447	
LWR-MOX-R											2.782		76.676	0.31		8.714
LWR-UOX											2.142		85.209	0.0436	0.447	

LWR-MOX											**2.782**		**76.676**	**0.31**		**8.714**
CAPRA											**2.146**		**9.972**	**0.263**		**7.483**
ADS											**1.337**		**0.126**	**1.617**		**7.675**
HTR-UOX													**28.287**	**0.0125**	**0.259**	
HTR-MOX										**37.688**			**3.929**	**0.281**		**3.715**

Table 2: Data matrix for some reactor types ("Matrix.xls").
UOX = open cycle, MOX = mixed U-Pu-oxide, CAPRA = fast Pu-burner, ADS = accelerator-driven system,
HTR = high temperature gas-cooled reactor
Post-fix "-R": Symbolises RESID-reactors (already existing at the beginning of the time horizon). In the case presented here the two types of LWR-R reactors have the same specific mass flow data as the LWR-reactors newly built within the time horizon.

Storage at first period:						
Th/U-out	**FP-out**	**U-depl-out**	**U-irr-out**	**MA-out**	**Pu-f-out**	**Pu-rec-out**
10000	2500	500000	100000	60	600	600
Uirr-Waste	**Pu-Waste**	**MA-Waste**				
0	0	0				
Loss factors:						
VPU:	0.001					
VUIRR:	0.001					
VMA:	0.001					

Table 3: An example of "Storage.xls": Includes initial values of 7 storages and 3 waste components (details see text above) as tonnes of materials or of heavy metals (HM) and loss factors for reprocessing of Pu, irradiated U and minor actinides (MA). The waste-storages here are set to zero.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
LWR-UOX-R	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90
LWR-MOX-R	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90
LWR-UOX	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90
LWR-MOX	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90
CAPRA	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90
ADS	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90
HTR-UOX	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90
HTR-MOX	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90

Table 4: An example of "capf.xls" for n = 2: 10 coarse and 20 small time steps. All the capacity factors are here set to the value of 0.90 (7884 h/year), as in Table 1. But basically the value for each reactor and time step can be set to any value independent from the values taken for the "coarse" time steps where the capacity factors are only dependent from the reactor type, but constant over time.

- "vleem2-input.xls" and "capf.xls" as "txt"-files:
  Apply "xls_to_txt.exe" to these 2 files and save as ".txt".
  Results are "vleem2-input.txt" and "capf.txt".
- "Martrix.xls" and "Storage.xls" as "csv"-files:
  Apply "xls_to_txt.exe" to these 2 files and save as ".csv".
  Results are "Matrix.csv" and "Storage.csv".

Additionally required are the executable programmes

- "vleem2_mass.exe" (for the nuclear mass flow computations with the "coarse" time step), and
- "vleem2_use.exe" (for the nuclear mass flow computations with the "small" time step, but based on the nuclear reactor structure of the coarse time steps).

These two programmes are the "source code" of the nuclear mass balance model NMBM.

3. The execution of the programme

The batch-file "execute.bat" steers the execution of the NMBM.

Table 5 shows its structure (programme):

```
set topic_path=%CD%
cd %topic_path%
del error_output.txt
del error_output_use.txt
vleem2_mass.exe vleem2_input.txt Matrix.csv Storage.csv
if not exist error_output.txt vleem2_use.exe vleem2_use_input.txt capf.txt
```

Table 5: The structure of "**execute.bat**".

There you can change the names of the input-files, but then the actual input-files must renamed accordingly ! Attention: The name "*vleem2_use_input.txt*" cannot be changed, because this file is produced internally during the run of **execute.bat**!

To start the execution
double "click" on the file "execute.bat":
Initiates the execution of

- "vleem2_mass.exe" In a successful operation two output EXCEL-files are produced:
  - output_cap.xls: Contains the information on the dynamics of various capacities for the coarse time steps.
  - output_mass.xls: Details of the dynamics of the nuclear mass flows, of the storages and the waste-volumes for the coarse time steps.

  An exe-file *"create_charts.exe"* is automatically called from *vleem2_mass.exe* and creates Excel-diagrams from the result tables so that the user gets graphical pictures (3 diagrams of capacities and 4 diagrams of nuclear mass flows) of the results with the "coarse" time steps:
  Reactor capacities:
  - Capacities divided according to the (three) reactor classes
  - Capacities of all reactor types
  - Capacity additions (new capacities) of all reactor types

Nuclear masses and mass flows:

- Temporal storage contents of minor actinides, fresh and multiple recycled Plutonium (tonnes)
- Consumption of natural Uranium (tonnes)
- Consumption of fresh Thorium (tonnes)
- Re-fabrication requirements (tonnes heavy metal/a)

The two files output_cap.xls and output_mass.xls are open and can be seen by pushing the icon "Microsoft Excel" in the task-bar. In this case you have to push the icon for **execute.bat** in the task-bar in order to proceed. If Excel is not open, then you can see the open file of **execute.bat**:
After successfully establishing these two files (they are open and must remain open at this intermediate stage of execution) **execute.bat** asks for the "number of short periods for JT": The number of "small" time steps within a "coarse" time step.
Put in this number and push the **enter** button.

If the structure of "capf.xls" corresponds precisely to this number **execute.bat** produces (without diagrams)

- output_use.xls: Details of the dynamics of the nuclear mass flows, of the storages and the waste-volumes for the small time steps.

- The quality of the results (for the coarse time steps) can be examined easily by controlling the diagrams in output_cap.xls and output_mass.xls:
  - If the results are **not satisfactory**, then close the two files output_cap.xls and output_mass.xls **without saving them.**
  - If the results are **satisfactory** and should be stored, then **save these two files under other names together with a change of the file-type from "*.txt" to "*.xls".**
- In the error case an error file "error_output.txt" or "error_output_use.txt" is produced, indicating the type of error. In such a case the corresponding EXCEL input-xls-file has to be opened, the correction be made, closed and saved (by applying "xls_to_txt.exe") as ".txt"- or ".csv"-file, respectively. "**execute.bat**" can then be re-started with a double-click.
- **Attention**:
  - Changes in an **input file** have always to be **saved!**
  - Before (re)starting "**execute.bat**" all output-files (output_cap.xls, output_mass.xls and output_use.xls) must be closed (otherwise they could not be overwritten with new results), **but not saved**.
  - The names of the output-files (output_cap.xls, output_mass.xls and output_use.xls) cannot be changed. For use under other names they have to be copied first and then renamed.
- If input-files with other names should be used, then they must be in the same file-domain as the batch-file "**execute.bat**", and in this batch-file the old names must be changed to the new names of the input-files **by applying the right mouse-button to the file "execute.bat" and then pushing "process" (German: "Bearbeiten")** (the next option behind "open"):
  ► See **Table 5**:
  - Line 5: If the files quoted above as "vleem2_input.xls" and/or "Matrix.xls" and/or "Storage.xls" get other names, then the corresponding ".txt"- and ".csv"-files in "**execute.bat**" must get the same names as these ".xls"-files.
  - Line 6: If the file quoted above as "capf.xls" gets another name, then "capf.txt" must get the same new name in "**execute.bat**".

- All other file-names must be kept in the original version:
  vleem2_mass.exe
  error_output.txt
  vleem2_use.exe
  vleem2_use_input.txt
- After having changed the file "**execute.bat**" press **"save"** before the next run of "**execute.bat**".

4. An important consistency check

The rationale of the main direction in long-term reactor development is the destruction of stockpiles of

- Plutonium (Pu),
- minor actinides (MA),
- some long-lived fission products (FP)
  (e.g., I-129, Cs-135, Tc-99, Sn-126, Se-79; realistic destruction capabilities, if at all, only for Iodine I-129 and Technetium Tc-99).

Table 2 (matrix of specific mass flow data for the considered reactors) shows that it contains also the specific charges and discharges of fresh Pu (Pu-f), multiple recycled Pu (Pu-rec), MA and FP (as a whole):

- Inputs: Columns 3 ,6, 7, 8
- Outputs: Columns 11, 14, 15, 16

Table 3 defines the storages (with initial values) for the FPs, MA, fresh Puu (Pu-f) and for the total Pu (Pu-rec, fresh + multiple recycled).
Currently there are no data for reactors destroying long-lived FPs by loading them from FP-storages, but Table 2 comprises reactors burning fresh and/or recycled Pu: All MOX-reactors, CAPRA and ADS. Only ADS is capable here of MA-destruction.

The main consistency checks are twofold:

4. The Pu-stockpiles must not be negative.
5. The total Pu-stockpile (Pu-rec) must not be smaller than the Pu-f stockpile.

If the Pu-stockpile results (last page of the EXCEL-file "output_cap.xls) of a run of "executive.bat" contradict these checks, then the capacities of newly built reactor types have to be changed accordingly in "vleem2-input.xls" (see Table 1 for the required consistency of corrections: if one capacity addition is raised, then another must be reduced by the same amount) in the lines below "Closure / new capacity in interval" in the periods before and/or in the same period where this contradiction appears the first time.
Prior to the new run of "execute.bat" do not forget to **save** and **close** the corrected "vleem2_input.xls", apply "xls_to_txt.exe" to this file and save it as "vleem2_input.txt"; and close "output_mass.xls", if it is still open. "

The input data presented in the Tables 1 to 4 above (for demonstrative purposes, although being more or less realistic for a "High Nuclear Case" for western Europe) produce "reason-able" results with regard to the two consistency checks quoted above, as shown below in Figure 1. Figures 2 and 3 show the corresponding capacity distributions and capacity additions. Time horizon is 2000-2100, with the "coarse" time steps of 10 years, i.e. 10 periods (see Table 1). The "small" time steps taken here are 5 years, i.e. 20 periods (see Table 4), but their results are not presented here. They are in accordance with the results of the coarse time steps (as can be easily checked by running "execute.bat" with the same data as above).

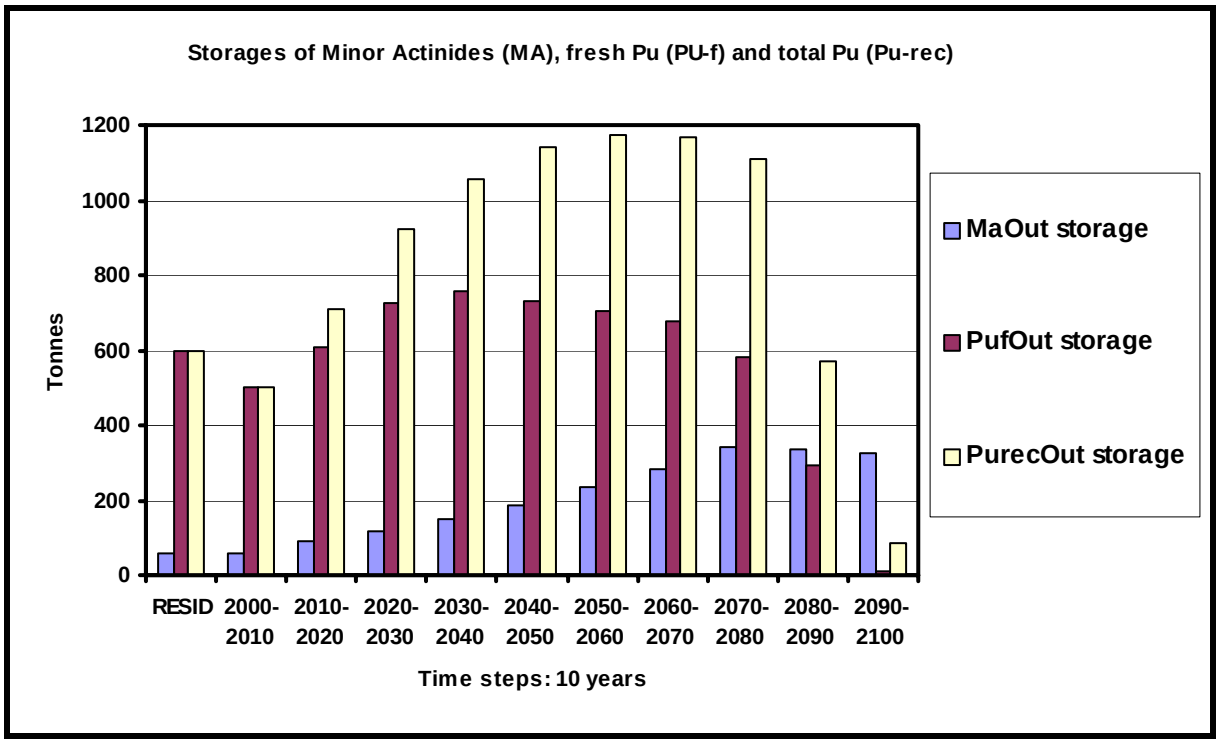


Figure 1: Dynamics of the storages for minor actinides – MA (blue), fresh (and once recycled) Pu – Puf (red), and total Pu – Purec (yellow).

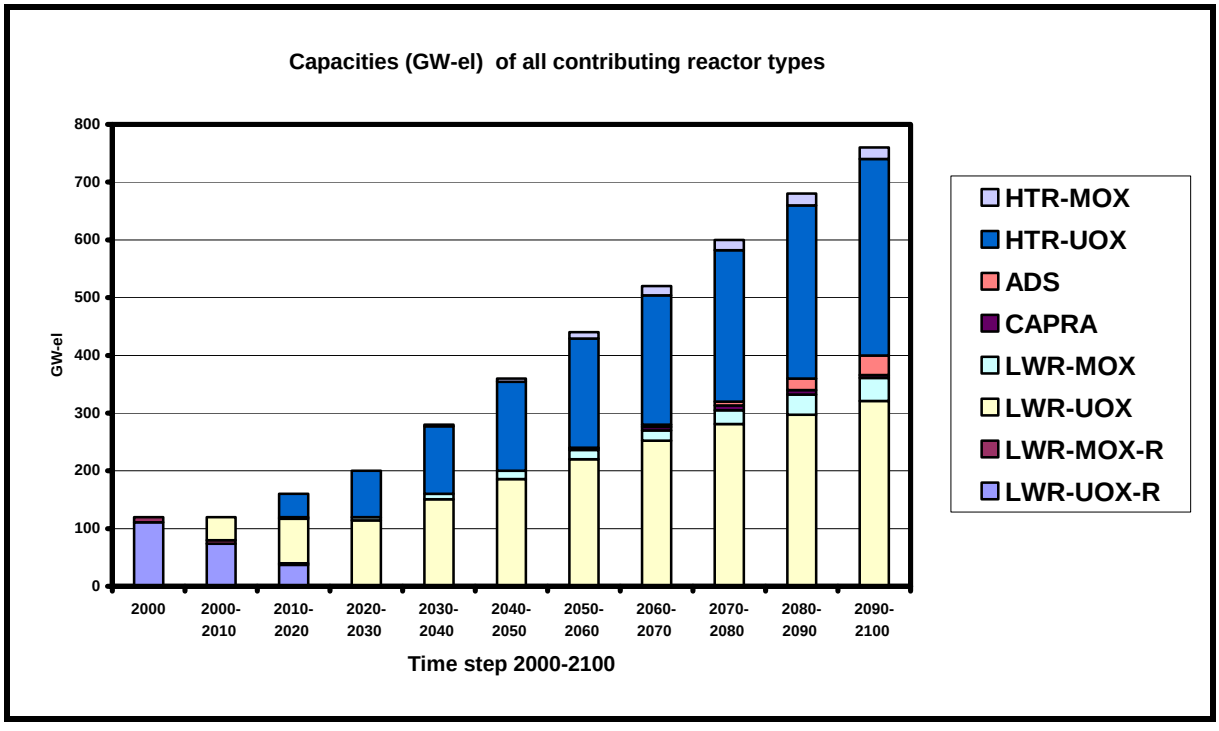


Figure 2: Dynamics of the capacities (GW-el) of the 9 contributing reactor types. Observe that the two RESID types (marked by "-R") can only die out, but not be added again.

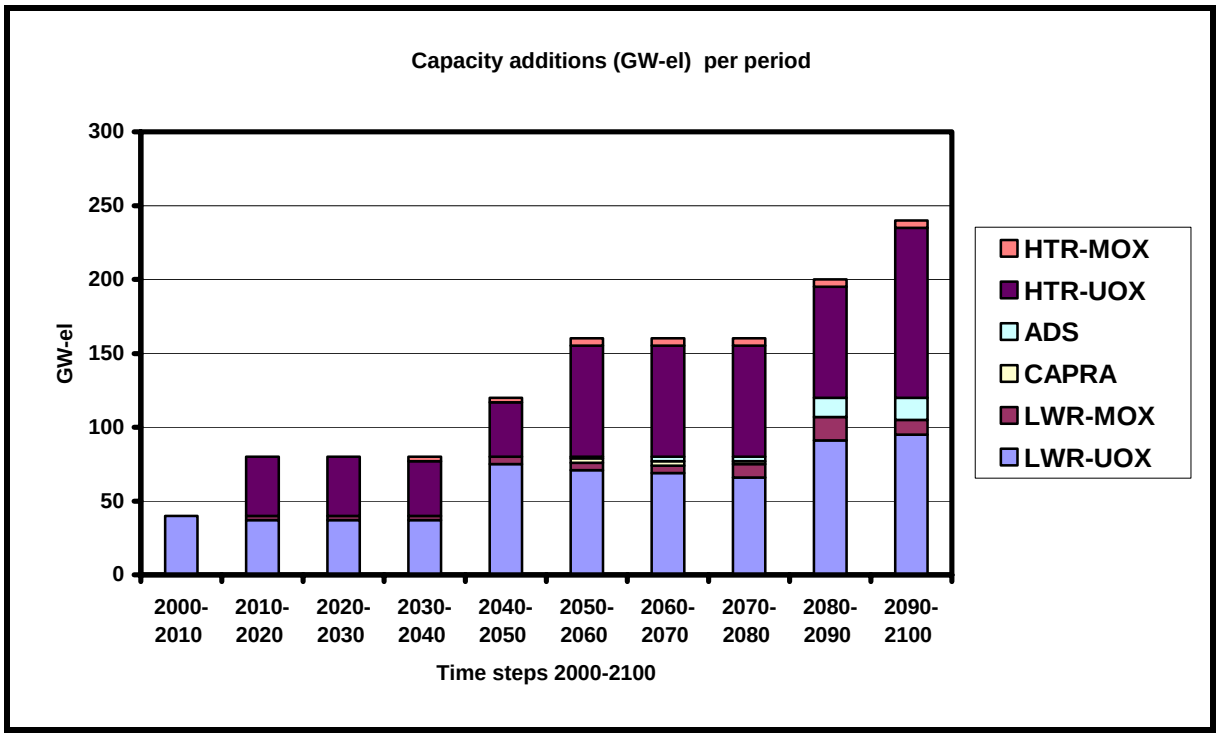


Figure 3: Dynamics of the capacity additions (GW-el per period) of the 7 contributing reactor types beyond the initially existing RESID-reactors. Here the specific data for the LWRs are independent from the membership to a reactor class.

Apparently it seems to be possible to reduce Pu-stockpiles even in cases of a pronounced nuclear expansion by extended additions of Pu-destroying reactors, but the reactor types included here do not reduce distinctly the MA stockpile. CAPRA and ADS are implemented first in the 6th period (2050-2060). It is not expected that these types can be commercially available earlier, due to the complex development steps for the reactors and their fuel cycles.

In Figures 2 and 3 it can be seen that the open-cycle types LWR-UOX and HTR-UOX remain dominating. The required fuel supply needs fresh U (and to a minor extent fresh Th). The reactor pool defined here cannot live alone from recycled fuel. There are two main reasons:

1. The capacity expansion is very pronounced.
2. No fast breeders are applied, only a fast reactor (CAPRA) as Pu-burner. A fast breeder pool could live quite well a long time from the stockpiles of depleted U (tails from enrichment plants). The lack of breeder reactors leads in this demonstration case also to a high consumption of natural U, as Figure 4 shows (next page):

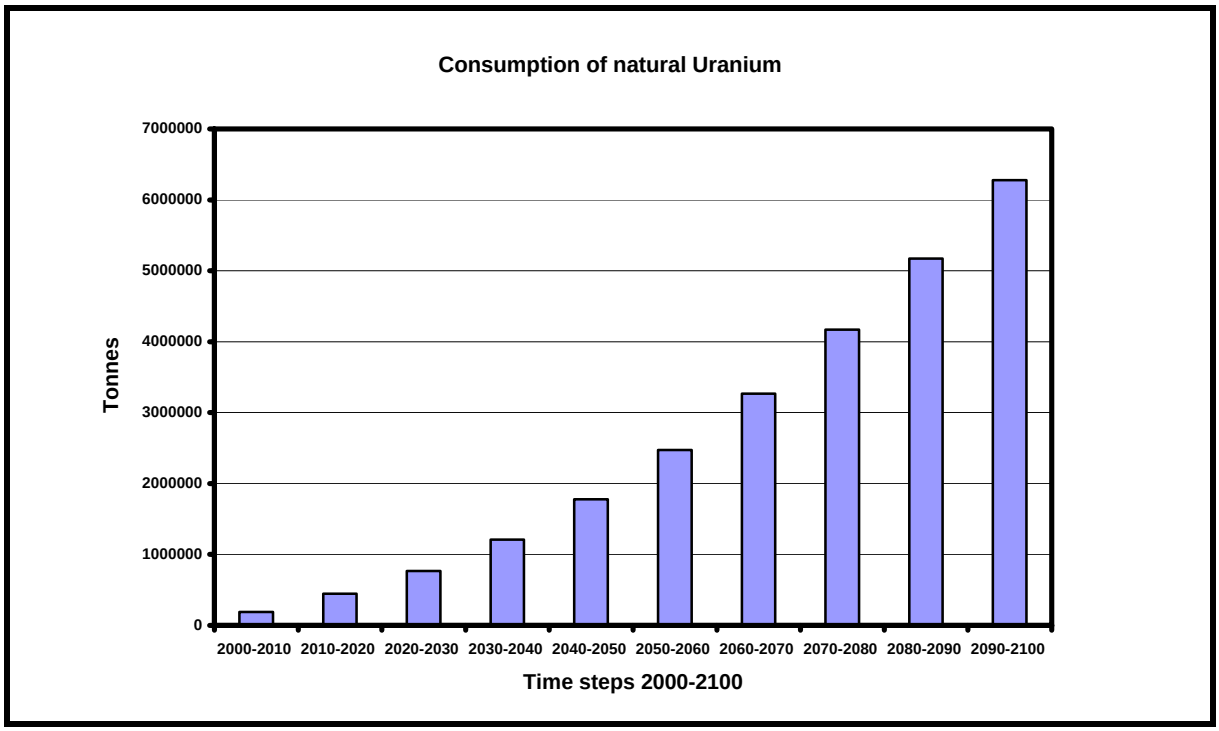


Figure 4: Dynamics of the natural-U consumption of the demonstration strategy. This case apparently leads to a very high U-consumption, nearly twice that much as there are known mineral U-reserves with production costs below 130 US$/kg U (~ 4 million tonnes).

Annex

The Nuclear Mass Balance Model (NMBM)

1. Introduction

The model consists of 3 parts (modules):

- The capacity programme
- The mass balance programme
- A programme to perform computations with deliberately small time steps based on the results of (more or less) coarse time steps of a size of (normally) some years, e.g., 5 or 10 years within a time horizon of some decades, say a 100 years.
  This 3rd module is mainly of interest,
  1. if the model is part of a comprehensive (superior) energy model with nuclear contributions, and
  2. if this energy model simulates the operation of a plant park in small time steps (months, weeks, days or less).

The NMBM presented below comprises therefore 3 time loops:

4. In the first loop the "coarse" time sequence of individual reactor capacities are calculated via the assumptions of individual capacity additions.
5. In the second loop the corresponding mass balances are calculated, considering operating capacities, capacity additions and retirements.
6. In the third loop the mass balances are calculated with deliberately small (constant) time steps and deliberately chosen load factors for each reactor type and time step (independent from the load factors in the "coarse" calculation where these factors are only reactor-dependent, not variable with time), but based on the dynamic reactor pool as defined for the "coarse" time steps (first and second loops).

2. Inputs for the capacity programme

2.1 Generic input parameters

Time horizon and loss factors of reprocessing:

First and last year of the time horizon (calendar years):	JI, JF	(e.g., 2000, 2100)
Time horizon (years):	JTH: JTH = JF – JI	(e.g., JTH = 100 years)
Length of time steps (years):	JT, divisor of JF – JI = JTH	(e.g., JT = 10 years)
Number of time steps:	IT = JTH / JT	(e.g., IT = 10)

Loss factors for the reprocessing of irradiated Uranium (U), Plutonium (Pu), and Minor Actinides (MA): VUIRR, VPU, VMA
Typically these loss factors are between 0.01 and 0.001.

In order to comply with the implemented formulas (see later) these values are corrected to
VUIRR = VUIRR/(1 + VUIRR) VPU = VPU/(1 + VPU) VMA = VMA/(1 + VMA)

It should be pointed out that the explanations below are mainly done or supported by the use of mathematical notation and FORTRAN-like statements for the sake of better understanding.

2.2 Nuclear capacities

2.2.1 Classes:

The capacities are divided into 3 "classes":

4. **RESID:** Nuclear plants (NP) already existing at the begin of the time horizon
5. **EL:** Electricity producers
6. **PH:** Process heat producers

Total capacity = RESID + EL + PH (GW-el)

All classes are expressed in electrical units: Gigawatt-electric – GW-el
Therefore the capacities are additive !

2.2.2 Definition of the total capacities for RESID, EL, PH

Prescribed for each time step I from I = 1 up to I = IT:
TCAP(Class, I): TCAP(RESID, I), TCAP(EL, I), TCAP(PH, I)

ToCAP(I) = TCAP(RESID, I) + TCAP(EL, I) + TCAP(PH, I) **total capacity**

2.2.3 Definition of reactor types

Separated according to the classes RESID; EL, PH: Type (Class)

Identification of the reactor types by their acronyms (LWR, HTR, etc. and further additions, like LWR1, LWR2, LWR-UOX, LWR-MOX, HTRold, HTRnew, HTR-PB /PB = pebble bed/).

Identification of the reactor types:

- Acronym of the name
- Plant life time: LT(Type(Class)): Years, must be a multiple of the time step JT
- Capacity factor: Full-load hours per year (year = 8760 h): FLH(Type(Class))
- Cooling times of the discharged fuel: ICT(Type(Class)): A multiple of the time step JT: 1, 2, 3, ….

Class RESID of the initially existing NPs:

- Number of reactor types and their names (acronyms)
- For each RESID-reactor type the decreasing capacity for each time step (until the capacity is – and remains – zero.
- The assumptions must fit into the prescribed time sequence of the total RESID-capacity !

Classes EL and PH as new NP-types:

- Number of reactor types and their names (acronyms)
- Time sequence of the total capacities for EL and PW: TCAP(Class, I)
- Time sequence of new plant additions for each Type(Class):
  CAPnew (Class, Type(Class); I) for each time step I
- **Control:**

$$TCAP_{Test}(Class, I) = TCAP(Class, I-1) + \Sigma\, CAPnew\ (Class, Type(Class), I) –$$

$$- \sum_{(Type(Class))} CAPnew\ (Class, Type(Class), I - LT(Type(Class)))$$

Diff (Class, I) = $TCAP_{Test}$(Class, I) – TCAP(Class, I)

Diff (Class, I) must be zero !

If DIFF (Class, I) > 0: Reduce CAPnew (Class, Type(Class), I)

If DIFF (Class, I) < 0: Enlarge CAPnew (Class, Type(Class), I)

Important: There are CAPnew-data only for the classes EL and PH !
The RESID-class is dying out !

2.2.4 Calculation of the individual capacities

If finally for all time steps from I = 1 to I = IT: DIFF(Class, I) = 0,
then the individual nuclear capacities can be calculated:

CAP(Class, Type(Class), I) =
= CAP(Class, Type(Class), I-1) + CAPnew(Class, Type(Class), I) –
– CAPnew(Class, Type(Class), I-LT(Type(Class)))

Retirements:
For RESID:
RET(RESID, Type(RESID), I) = CAP(RESID, Type(RESID), I-1) – CAP(RESID, Type(RESID), I)
For EL and PH:
RET(Class, Type(Class), I) = CAPnew(Class, Type(Class), I-LT(Type(Class)))

Electricity- and process heat production of each plant during time step I:
PRO = "product", expressed in electrical Terawatt-hours (TWH-el)

PRO(Class, Type(Class), I) = CAP(Class, Type(Class), I)∗FLH(Type(Class))∗JT (single plant)

$$CPRO(Class, I) = \sum_{(Type(Class))} PRO(Class, Type(Class), I)$$ all plants of a Class

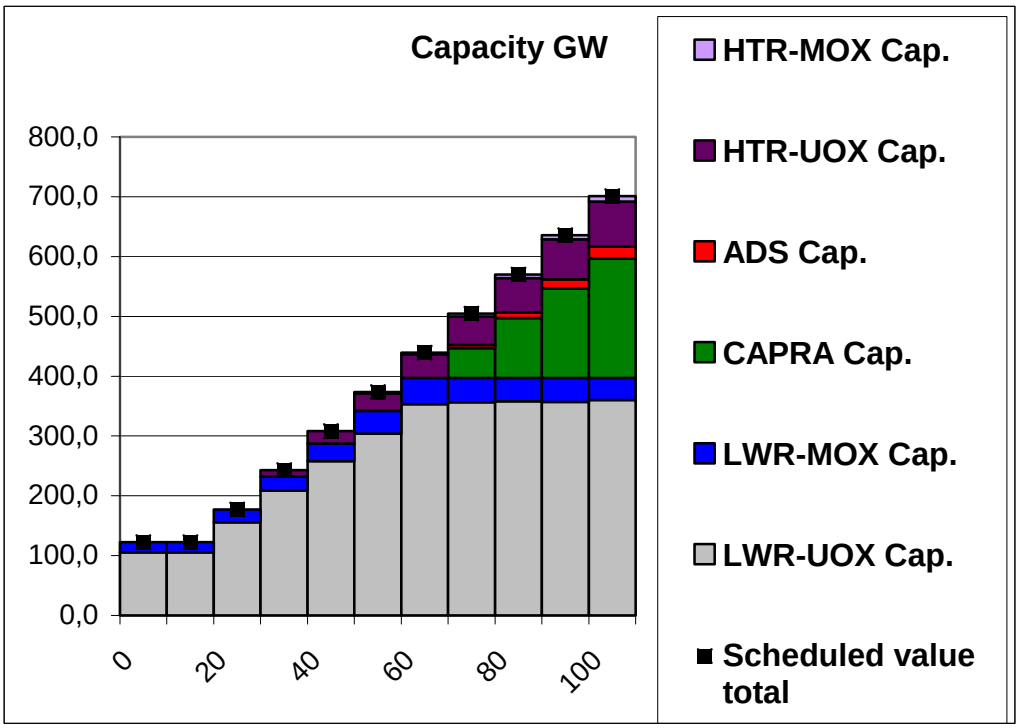
$$ToPRO(I) = \sum_{Class} CPRO(Class, I)$$ all plants

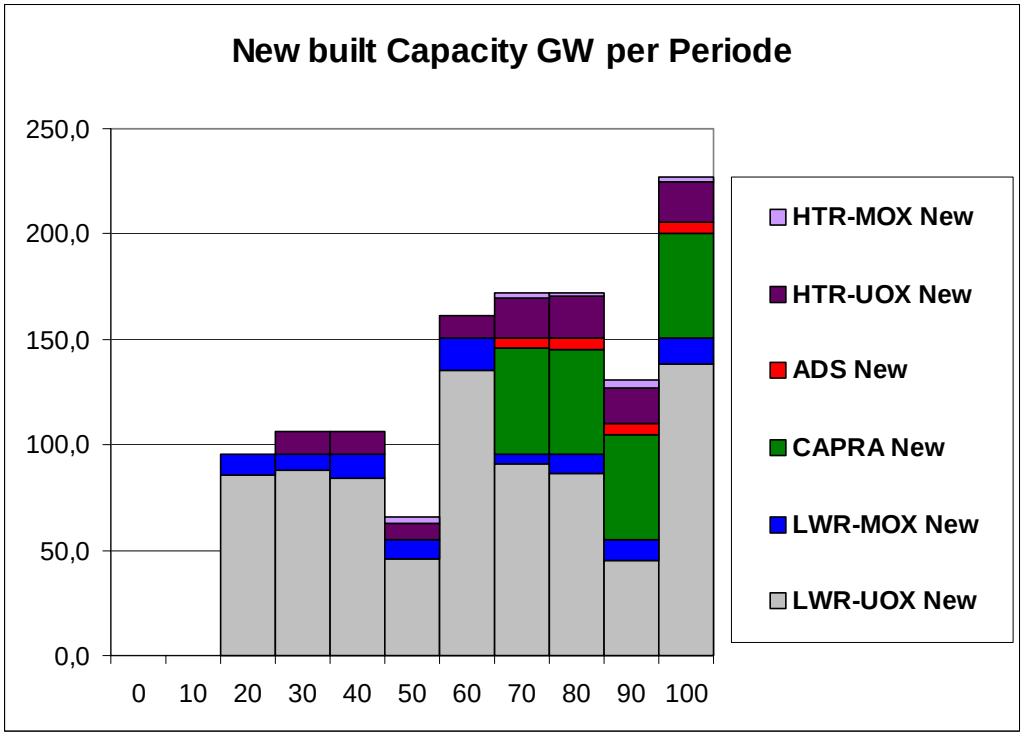
The computations are done for all time steps from I = 1 to I = IT:
Results are time series of all

- capacities,
- new installations,
- expirations,
- electricity and process heat productions (expressed in TWh_{el}).

These results allow the computation of the nuclear mass flows.

The diagrams on this page show for the sake of clarity the capacity results as derived with the former EXCEL-programme of VLEEM1:
Inputs were the total capacities for electricity and process heat (the latter for the HTRs, expressed as GW_{el}), and then the capacity additions for each reactor type to fill up the prescribed total capacities.





3. Computations of the nuclear mass flows

3.1 The nuclear fuel cycle

The nuclear fuel cycle is divided into 5 steps (Figure 3.1):

- Fresh heavy metals (HM): Natural Uranium – U_{nat}, and Thorium – Th
- Fuel element-fabrication - fresh and recycled ones ("re-fabrication")
- Reactor operation
- Interim storage for spent fuel elements, then possibly reprocessing and re-fabrication
- Final repository

Figure 6.1. **Generic nuclear fuel cycle model showing the essential elements and options for a nuclear fuel cycle**

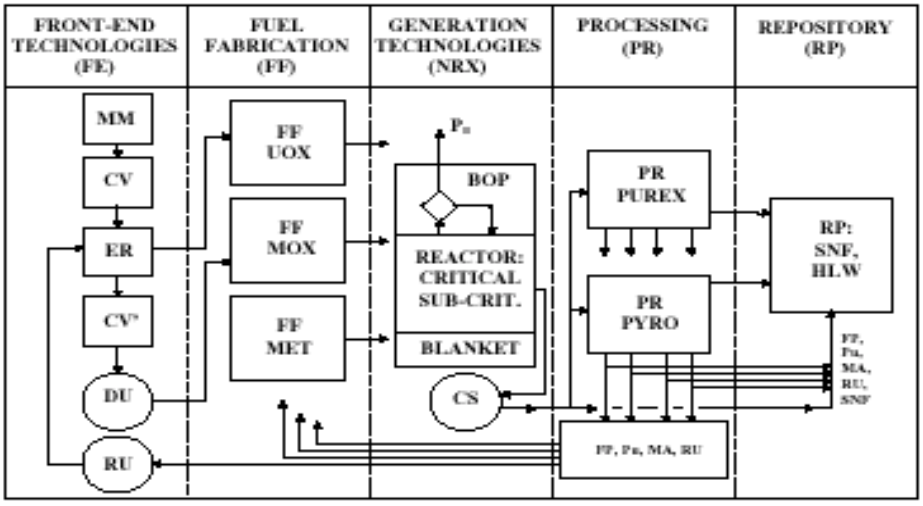


Figure 6.2. **Essential elements of the nuclear fuel cycle (NFC) being considered in evaluating key sustainability metrics for nuclear energy**

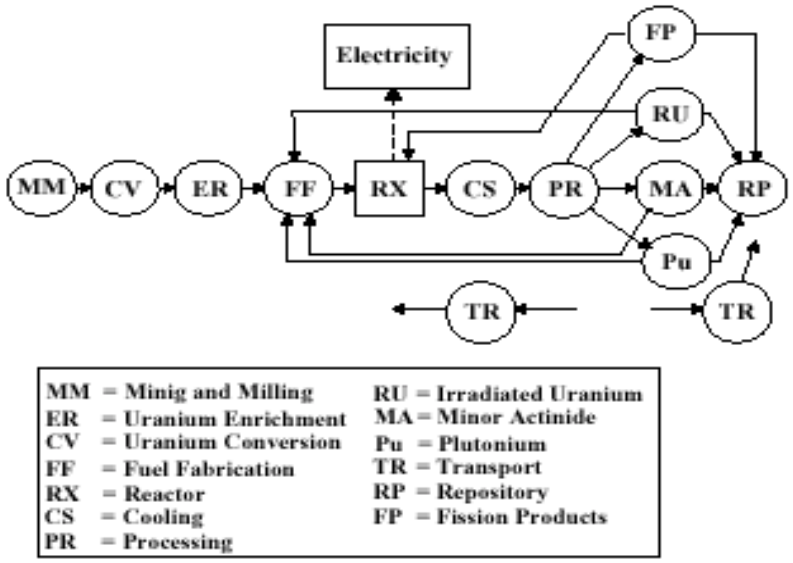


Figure 3.1: Generic nuclear fuel cycle model (upper part) and the essential elements of the nuclear fuel cycle (lower part): DU = depleted U, MET = metal fuel /NEA 2002/

The nuclear fuel cycle supplies not only one reactor or one reactor type, but an entire, more or less coupled system of reactor types, schematically shown in Figure 3.2:

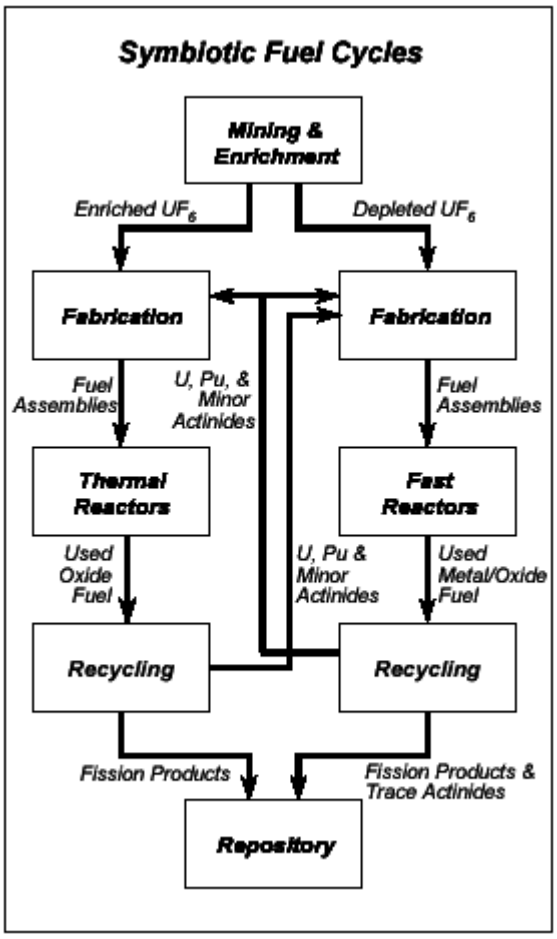


Figure 3.2: Scheme of a symbiotic fuel cycle comprising the interference between thermal and fast reactors (recycle of actinides from thermal into fast reactors) /NERAC 2002/

Each of the reactors discharges into each interim storage (possibly also zero) and can take material from each storage (via re-fabrication, possibly also zero).
The numerical description goes via 8 "pots" and the final disposal:
Fresh fuel requirements **(per time step and cumulated):**

- U_{nat}
- Th

Interim storages:

- Fission products (FP),
- Depleted U (U-depl),
- Irradiated U (U-irr),
- Minor actinides (MA)
- Fresh (once recycled) Pu (Pu-f)
- Multiple recycled Pu (Pu-rec)

Final disposal:

- Irradiated (spent) Th and U destined for repository (Th/U-out)
- Waste: Pu and MA (entire discharges or only their re-fabrication losses), FP, U, Th

For the interim storages initial values ("RESIDs") must be set (see Chapter 3.4 below).

In the case of fresh fuel the inputs require the amount of fresh natural material: U_{nat} and Th. The enrichment-value is not used explicitly in this model (see Chapter 3.2).

Fuel fabrication:

- Fresh fuel (U_{nat} and Th)
- MOX-re-fabrication: Mixed-oxide fuel made of U and/or Th and Pu
- Re-fabrication for fast reactors (FR): Material from all interim storages, possibly also fresh HM
- ADS-re-fabrication: Material from all interim storages, possibly also fresh HM

3.2 Description of the individual reactor types

Table 3.1 shows the description matrix for the reactors contributing to a scenario computation, here defining 3 classes, each with 2 reactor types. These classes are only of relevance for the computation of capacities (operating, new and retiring ones). For the mass flow computations this reference to classes is skipped, because here being irrelevant (see below).

Each reactor type has 3 data blocks:

- First core (t/GW-el)
- Rest core (t/GW-el)
- "Reloads" (charges) and discharges ($T/TW_{el}h$)

Each reactor has 16 components for these 3 items:

- Columns 1 – 9 for charges,
- Columns 10 – 16 for discharges.

Matrix elements without a number are set to zero.

Charges (columns 1 to 9, called "Reload") ($t/TW_{el}h$):

1. U-nat-in: Natural U calculated according to the required enrichment of fresh U-fuel.
2. Th-in: Charge of fresh Th.
3. FP-in: Charge with fission products (FP)
4. U-depl-in: Charge with depleted U
5. U-irr-in: Charge with recycled U
6. MA-in: Charge with minor actinides (MA)
7. Pu-f-in: Charge with "fresh" (only once reprocessed) Pu
8. Pu-rec-in: Charge with multiple recycled Pu
9. HM-in: Charge with heavy metal containing reprocessed material (for the computation of re-fabrication requirements)

Discharges (columns 10 to 16 of "Reload") ($t/TW_{el}h$):

10. Th/Pu-out: Discharge of irradiated Th and/or U for final disposal
11. FP-out: Discharge of fission products (FP)
12. U-depl-out: Tails of fresh enriched U ("depleted" U), "waste" of U235-enrichment
13. U-irr-out: Discharge of irradiated U
14. MA-out: Discharge of MA
15. Pu-f-out: Discharge of "fresh" Pu (originating from fresh U)
16. Pu-rec-out: Discharge of recycled Pu

Table 3.2 shows a data matrix of reactor types with data (except zeros in empty spaces), without reference to classes.

Column number:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	U-nat-in	Th-in	FP-in	U-depl-in	U-irr-in	MA-in	Pu-f-in	Pu-rec-in	HM-in	Th/U-out	FP-out	U-depl-out	U-irr-out	MA-out	Pu-f-out	Pu-rec-out
Reload t/TWh																
Class 1: NPP-Type 1																
NPP-Type 2																
Class 2: NPP-Type 3																
NPP-Type 4																
Class 3: NPP-Type 5																
NPP-Type 6																
First core t/GW																
Class 1: NPP-Type 1																
NPP-Type 2																
Class 2: NPP-Type 3																
NPP-Type 4																
Class 3: NPP-Type 5																
NPP-Type 6																
Rest core t/GW																
Class 1: NPP-Type 1																
NPP-Type 2																
Class 2: NPP-Type 3																
NPP-Type 4																
Class 3: NPP-Type 5																
NPP-Type 6																

Table 3.1: Schematic data matrix for 6 reactor types with formal identification of their relationships to the 3 classes.
NPP = nuclear power plant.
There must be a number in each matrix element, or zero, because sums are running along rows and columns.

Column number:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	U-nat-in	Th-in	FP-in	U-depl-in	U-irr-in	MA-in	Pu-f-in	Pu-rec-in	HM-in	Th/U-out	FP-out	U-depl-out	U-irr-out	MA-out	Pu-f-out	Pu-rec-out
Reload t/TWh																
LWR-OC	20.513										0.123	18.119	2.2385	0.0036	0.0288	
LWR-MOX				2.204			0.194		2.398		0.122		2.128	0.018		0.1322
CAPRA-MOX				0.050	0.2595			0.2475	0.557		0.1225		0.2598	0.015		0.1797
TRU-burner				0.0543	0.4688	0.0307	0.0489	0.1788	0.7815		0.1088		0.4692	0.0245		0.179
FR (CR = 1)	0.1128				1.9328	0.0056		0.1816	2.2328		0.1124		1.933	0.0056		0.1818
TRU-ADS					0.0056	0.0786		0.3953	0.4795		0.1192		0.0056	0.0656		0.2811
First core t/GWe																
LWR-OC	539.10											476.20				
LWR-MOX				57.90			5.10		63.0							
CAPRA-MOX				1.590	8.254			7.873	17.717							
TRU-burner				2.032	17.547	1.149	1.830	6.693	29.251							
FR (CR = 1)	4.231				72.491	0.210		6.811	83.743							
TRU-ADS					0.124	1.742		8.763	10.629							
Rest core t/GWe																
LWR-OC											2.04		76.87	0.06	0.478	
LWR-MOX											2.02		71.78	0.298		5.405
CAPRA-MOX											2.146		9.972	0.263		7.483
TRU-burner											2.188		19.959	1.110		8.180
FR (CR = 1)											0.763		27.007	0.076		2.467
TRU-ADS											1.337		0.126	1.617		7.675

Table 3.2: Data matrix for some reactor types. OC = open cycle, MOX = mixed U-Pu-oxide
TRU = excess transuranics from fast breeders, FR = fast reactor, CR = conversion ratio (of U238 to Pu)
CAPRA = fast Pu-burner, ADS = accelerator-driven system

Definition: **Types ≡ number of all reactor types**

The data structure is of the mode
MATRIX(rows, columns) = MATRIX(3 x number of all reactor types, MAT) =
= MATRIX(3 x Types, MAT)
Rows of MATRIX from 1 to 3 x Types:

Type = 1 to Types:	Rows related to "Reload"
Type = Types+1 to 2 x Types:	Rows related to "First core"
Type = 2 x Types +1 to 3 x Types:	Rows related to "Rest core"

Columns of MATRIX from MAT = 1 to MAT = 16:

MAT = 1 to 9:	Specific material inputs (charges) into the reactors
MAT = 10 to 16:	Specific material outputs (discharges) of the reactors

3.3 Formulas for the specific reactor charges

The reactor types are now numbered from 1 to "Types" without further reference to classes !

Specific values MATIN(MAT; I) for the nuclear substances (materials) MAT = 1 to 9 for the time step I = 1 to IT:

The formulas are written in a FORTRAN-similar notation, but the model is written in C++ !

The basic formula for re-fabrication sums only over reactors "Refab-Type" using re-fabricated material (CAPnew is the capacity (GWe) of reactor type "Refab-Type" newly installed per time step I, CAP is the capacity of reactors type "Refab-Type" (GWe) operated within the period I under consideration):

Refab (t HM/a, I) =

$$= \{ \sum_{\text{Re } fab-Type} \textbf{CAPnew(Refab-Type, I) x HM-loading}_{\textbf{first core}}\textbf{(Refab-Type)}\}/JT +$$

$$+ \sum_{\text{Re } fab-Type} \textbf{CAP(Refab-Type, I) x HM-loading(Refab-Type) per annum } \}$$

MAT =1:9 Inputs into the individual reactor types:

$$MATIN(MAT,I) = \sum_{Type=1}^{Types} PRO(Type, I) * MATRIX(Type, MAT) +$$

$$+ \sum_{Type=1}^{Types} CAPnew(Type, I) * MATRIX(Types+Type, MAT)$$

```
Do I = 1 to IT
 Do MAT = 1 to 9
  MATIN(MAT, I) = 0
  Do Type = 1 to Types
   TMATIN1(Type, MAT, I) = PRO(Type, I) * MATRIX(Type, MAT)
   TMATIN2(Type, MAT, I) = CAPnew(Type, I) * MATRIX(Types + Type, MAT)
   MATIN(MAT, I) = MATIN(MAT, I) + TMATIN1(Type, MAT, I) + TMATIN2(Type,
   MAT, I)
  end Type
 end MAT
```

```
 Refab(I) = MATIN(9, I)/JT          Annual re-fabrication amounts (t HM/a) in time
                                    step I
 UNAT(I) = MATIN(1, I)              Unat-requirements in period I (t/JT)
end I
UNATT(1) = UNAT(1)
THT(1) = MATIN(2, 1)

Do I = 2 to IT
 UNATT(I) = UNATT(I-1) + UNAT(I),   Cumulated Unat-requirements up to period I
                                    (tonnes t)
 THT(I) = THT(I-1) + MATIN(2, I),   Cumulated Th-requirements up to period I
                                    (tonnes t)
end I
```

3.4 Formulas for the specific reactor discharges

Materials in storages:
Formulas for STORAGE(MAT, I) and TMATOUT(MAT, I) for the substances MAT = 10 to 16 in the time steps I = 1, 2, ... ,IT

MAT = 10:16 **Material outputs from the individual reactor types:**

```
Do I = 1 to IT
 Do MAT = 10 to 16
  Do Type = 1 to Types
   TMATOUT1(Type, MAT, I) = PRO(Type, I) ∗ MATRIX(Type, MAT)
   TMATOUT2(Type, MAT, I) = Still(Type, I) ∗ MATRIX(2∗Types + Type, MAT)
   TMATOUT(Type, MAT, I)=TMATOUT1(Type, MAT, I) + TMATOUT2(Type, MAT, I)
  end Type
 end MAT
end I
```

Balance of the storages for FPs, U-depl, U-irr, MA, PU-f and total Pu (MAT = 10 to 16):

STORAGE(MAT, 1) = RESID-values (initial values) for MAT = 10 to 16

STORAGE(16, I) = STORAGE(Pu-total: Fresh or once recycled plus multiple recycled Pu, I)

```
Do I = 1 to IT
 MAT = 10:15 !!
 Do MAT = 10 to 15
  FACTOR = 0
  Do Type = 1 to Types
   If (I-ICT(Type)) > 0 then FACTOR = FACTOR + TMATOUT(Type, MAT, I-ICT(Type))
  end Type
  if I = 1 then STORAGE(MAT, 1) = STORAGE(MAT, 1) + FACTOR
  else STORAGE(MAT, I) = STORAGE(MAT, I-1) + FACTOR
  If MAT > 10 then STORAGE(MAT, I) = STORAGE(MAT, I) – MATIN(MAT-8, I)
 end MAT
```

```
 MAT = 16:
 if I = 1 then STORAGE(16, 1) = STORAGE(16, 1) + FACTOR – MATIN(7, 1)
 else STORAGE(16, I) = STORAGE(16, I-1) + FACTOR – MATIN(7, I)
 FACTOR = 0
  Do Type = 1 to Types
   If (I-ICT(Type)) > 0 then FACTOR = FACTOR + TMATOUT(Type, 16, I-ICT(Type))
  end Type
 STORAGE(16, I) = STORAGE(16, I) + FACTOR – MATIN(8, I)
end I
```

3.5 Computation of the waste-amounts

The waste comprises the cumulated discharged (spent) amounts of

- Th, FPs, non-re-fabricated amounts of U, MA, and PU, and
- re-fabrication losses of losses of U, Pu, and MA

```
UIRRwaste(1) = RESID-value
PUwaste(1) = RESID-value
MAwaste(1) = RESID-value
```

VPU, VUIRR, VMA: Loss factors at reprocessing of U-irr, Pu and MA (see first page)

```
Do I = 1 to IT
 If I = 1 then do
  UIRRwaste(1) = UIRRwaste(1) + VUIRR∗MATIN(5, 1)
  PUwaste(1) = PUwaste(1) + VPU∗(MATIN(7, 1) + MATIN(8, 1))
  MAwaste(1) = MAwaste(1) + VMA∗MATIN(6, 1)
 end if
 else do
  UIRRwaste(I) = UIRRwaste(I-1) + VUIRR∗MATIN(5, I)
  PUwaste(I) = PUwaste(I-1) + VPU∗(MATIN(7, I) + MATIN(8, I))
  MAwaste(I) = MAwaste(I-1) + VMA∗MATIN(6, I)
 end else
 WASTE(I) = STORAGE(10, I) + STORAGE(11, I) + UIRRwaste(I) + PUwaste(I) +
          + MAwaste(I)
end I
```

List of the required RESID-values (initial values at the beginning of the first period):

```
STORAGE(MAT, 1) = RESID-values for MAT = 10 to 16
UIRRwaste(1) = RESID-value
DPUwaste(1) = RESID-value
DMAwaste(1) = RESID-value
```

4. Computations with deliberately small time steps

4.1 Generic aspects

These computations are based on the reactor structure as defined above with the “coarse” time step, but proceed along small constant time steps and freely fixable capacity (load) factors,

separately for each reactor type and time step. Above, for the computations with the coarse time steps, the capacity factors were only reactor type dependent, but constant in time: FLH(Type).

These "use"-computations run with deliberately short constant time steps, as integer fraction of the coarse step JT:

⇒ IN Number of short time steps in JT
⇒ N = IT∗IN Total number of the small time steps:
J = 1 to N

BT = JT/IN length of the small time step (a), e.g., ¼ [year]
Time unit is [year].

The load CAPF(Type, J) = capacity factor of the reactor „Type" in the small time step J, is an input data in this module, but also can be transferred from external programmes:
CAPF(Type = 1 to Types, J =1 to N) [0 ≤ CAPF ≤ 1

4.2 Computation of the mass balances for the individual reactor types

Definition of matrix dimensions:
DSTORAGE(MAT = 10 to 16, J = 1 to N)
DMATIN(MAT= 1 to 9, J = 1 to N)
DUNAT(J = 1 to N)
DUNATT(J = 1 to N)
DTHT(J = 1 to N)
DPUwaste(J = 1 to N), DMAwaste(J = 1 to N), DUIRRwaste(J = 1 to N),
DWASTE(J = 1 to N)

```
TMATIN1 = TMATIN1/IN
TMATIN2 = TMATIN2/IN
TMATOUT1 = TMATOUT1/IN
TMATOUT2 = TMATOUT2/IN

DUNATT(1) = 0
DTHT(1) = 0

Do J = 1 to N
 Do Type = 1 to Types
  CAPF(Type, J) = CAPF(Type, J)*8.76/FLH(Type)
 end type
I = 1 + (J-1)/IN
 Do MAT = 1 to 9
  DMATIN(MAT, J) = 0
  Do Type = 1 to Types
   DMATIN(MAT, J) = DMATIN(MAT, J) + TMATIN1(Type, MAT; I)∗CAPF(Type, J) +
   + TMATIN2(Type, MAT; I)
  end Type
 end MAT
 if J = 1 then do
  DUNATT(1) = DUNATT(1) + DMATIN(1, 1)
  DTHT(1) = DTHT(1) + DMATIN(2, 1)
 end if
```

```
 else do
  DUNATT(J) = DUNATT(J-1) + DMATIN(1, J)
  DTHT(J) = DTHT(J-1) + DMATIN(2, J)
 end else
```

REFABR(J) = DMATIN(9, J)/BT	Annual re-fabrication amount (t HM/a) in period J
DUNAT(J) = DMATIN(1, J)	Consumption of natural U (U_{nat}) in period J
DUNATT(J)	Cumulated consumption of U_{nat} up to period J (tonnes)
DTHT(J)	Cumulated consumption of Thorium up to period J (tonnes)

```
 Do MAT = 10 to 15
  FACTOR = 0
  Do Type = 1 to Types
        If (I-ICT(Type)) > 0 then FACTOR = FACTOR +
        + TMATOUT1(Type, MAT, I-ICT(Type)) ∗CAPF(Type, J) + TMATOUT2(Type,
          MAT; I-ICT(Type))
  end Type
  if J = 1 then DSTORAGE(MAT, 1) = DSTORAGE(MAT, 1) + FACTOR
  else DSTORAGE(MAT, J) = DSTORAGE(MAT, J-1) + FACTOR
  If MAT > 10 then DSTORAGE(MAT, J) = DSTORAGE(MAT, J) – DMATIN(MAT-8, J)
 end MAT
```

MAT = 16:

```
 if J = 1 then DSTORAGE(16, 1) = DSTORAGE(16, 1) + FACTOR – DMATIN(7, 1)
 else DSTORAGE(16, J) = DSTORAGE(16, J-1) + FACTOR – DMATIN(7, J)
 FACTOR = 0
  Do Type = 1 to Types
        If (I-ICT(Type)) > 0 then FACTOR = FACTOR +
        + TMATOUT1(16, I-ICT(Type)) ∗CAPF(Type, J) + TMATOUT2(16, I-ICT(Type))
  end Type
 DSTORAGE(16, J) = DSTORAGE(16, J) + FACTOR – DMATIN(8, J)
end J
```

4.3 Computation of the waste amounts

The waste comprises (as above for the coarse time step) the cumulated discharged (spent) amounts of

- Th, FPs, non-re-fabricated amounts of U, MA, and PU, and
- re-fabrication losses of losses of U, Pu, and MA

```
DUIRRwaste(1) = RESID-value
DPUwaste(1) = RESID-value
DMAwaste(1) = RESID-value
```

VPU, VUIRR, VMA [1]: Loss factors at reprocessing of U-irr, Pu and MA (see first page)

```
Do J = 1 to N
 If J = 1 then do
   DUIRRwaste(1) = DUIRRwaste(1) + VUIRR∗DMATIN(5, 1)
   DPUwaste(1) = DPUwaste(1) + VPU∗(DMATIN(7, 1) + DMATIN(8, 1))
   DMAwaste(1) = DMAwaste(1) + VMA∗DMATIN(6, 1)
 end if
 else do
   DUIRRwaste(J) = DUIRRwaste(J-1) + VUIRR∗DMATIN(5, J)
   DPUwaste(J) = DPUwaste(J-1) + VPU∗(DMATIN(7, J) + DMATIN(8, J))
   DMAwaste(J) = DMAwaste(J-1) + VMA∗DMATIN(6, J)
 end else
 DWASTE(J) = DSTORAGE(10, J) + DSTORAGE(11, J) + DUIRRwaste(J) + DPUwaste(J)
                + DMAwaste(J)
end J
```

List of the required RESID-values (initial values at the beginning of the first period):

DSTORAGE(MAT, 1) = STORAGE(MAT, 1) = RESID-values for MAT = 10 to 16
DUIRRwaste(1) = UIRRwaste(1) = RESID-value
DPUwaste(1) = PUwaste(1) = RESID-value
DMAwaste(1) = MAwaste(1) = RESID-value

References:

/NEA 2002/: NEA-OECD: Accelerator-driven Systems (ADS) and Fast Reactors (FR) in Advanced Nuclear Fuel Cycles, A Comparative Study, OECD 2002, http://www.nea.fr/html/pub/webpubs/welcome.html#ndd

/NERAC 2002/: Nuclear Energy Research Advisory Committee: A Technology Roadmap for Generation IV Nuclear Energy Systems, Technical Roadmap Report, December 2002, GIF-002-00, http://www.nuclear.gov/

Part 3: Operational Users´ Guide for the PC-based Nuclear Mass Flow Programmes

VLEEM2_MASS
(Capacity- and Mass Balance-Computations)
and
VLEEM2_USE
(Operational Mass Balance Computations)

Sylvia Gasper, Gerhard Kolb

November 2003

1. Introduction

VLEEM2_MASS: Allows dynamic capacity- and mass flow computations for a nuclear reactor park with "coarse" constant time steps (e.g., 10 years) within a deliberately large time horizon (e.g., 100 years).

VLEEM2_USE: Produces mass flow computations with deliberately small constant time steps, but with variable load factors, based on the input data and results of VLEEM2_MASS: It operates with the same reactor configurations.

The explanations below refer only to the **practical** execution of the two modules of the **Nuclear Mass Balance Model (NMBM)**, after its input has been prepared as briefly explained in the NMBM description in the **Annex.**

This input includes mainly:

- The time horizon (e.g., 2000-2100)
- Size of the "coarse" time step (e.g., 10 years, i.e. 10 time steps)
- 3 "classes" of reactors:
  7. RESID: Nuclear plants (NP) already existing at beginning of the time horizon
  8. EL: Electricity producers
  9. PH: Process heat producers

  All classes are expressed in electrical units: Gigawatt-electric – GW-el
  Total capacity = RESID + EL + PH (GW-el) in each time step
- Definition of the reactor types for each class
- For each RESID-reactor type the decreasing capacity for each time step (until the capacity is – and remains – zero).
- Time sequence of the total capacities for EL and PW
- Time sequence of new plant additions for each type in each class [Type(Class)]: CAPnew (Class, Type(Class); I) for each time step I: Must be conform to the time sequence of the total capacities for EL and PW ! (Is checked by the PC-programme !)

These data are basic definition items of the scenarios and are organised in an EXCEL-input file, here called "vleem2-input". An example of it is shown in Table 1, including explanations of the data contents.

Additionally the following reactor-specific data are required:

- Each reactor type has 3 data blocks:
  - First core (t/GW-el)
  - Rest core (t/GW-el)
  - "Reloads" (charges) and discharges (t/TW_{el}h)

  Each reactor has 16 components for these 3 items:
  - Columns 1 – 9 for charges,
  - Columns 10 – 16 for discharges.

  Matrix elements without a number are set to zero.

Table 2 shows a data matrix of reactor types (to be prepared as EXCEL-file "Matrix.xls") with data (except zeros in empty spaces), without reference to classes, except the RESID-reactors (already existing at the beginning of the time horizon), marked with the post-fix "-R". In the case presented here the two types of LWR-R reactors have the same specific mass flow data as the LWR-reactors newly built within the time horizon. The reactors in Table 2 must be ordered in the same sequence as in Table 1.

Furthermore the following information items are needed:

- Initial values for storages (EXCEL-file "Storage.xls") (in tonnes heavy metal – t HM) of Th/U (discharges for final disposal), fission products (tonnes of fission products – t FP), depleted U (U-depl) from enrichment plants, irradiated U (U-irr – for potential reuse), minor actinides (MA), fresh (i.e. not or once recycled) Pu, and total Pu (fresh and multiple-recycled Pu). Storage.xls contains also loss-factors for reprocessing of Pu, irradiated U (U-irr) and MA. Target values for these latter items are less than 1 %, near to 0.001 (Table 3).
- EXCEL file "capf.xls": Capacity factors for computations with "small" time steps (integer fraction n of the "coarse" time steps) with

  $$0 \leq \text{capf} \leq 1$$

  for each reactor type (same sequence as in Table 1) and small time step.
  Table 4 shows such a matrix "capf.xls" for the case of 10 coarse and 20 small time steps (n = 2). The matrix must always correspond exactly to the total number of small time steps (columns = n x number of coarse time steps), not less and also not more! The value for each reactor and time step can be set to any value independent from the values taken for the "coarse" time steps where they are only dependent from the reactor type, but constant over time (and expressed in hours per year).

2. Requirements for the execution of the programme

Required input files:

- EXCEL-file "vleem2-input" of the basic scenario input data (Table 1)
- EXCEL-file "Matrix.xls" with the specific mass flow data for the reactors (Table 2)
- EXCEL-file "Storage.xls" with initial values for 7 storages and 3 loss factors for re-processing (Table 3)
- EXCEL-file "capf.xls" with capacity factors for computations with "small" time steps (Table 4)

These 4 EXCEL-files, if required, are to be changed in EXCEL, but must be saved prior the first execution and after each data change as ".txt"- or ".csv"-file:

Year of beginning:	2000									
Year of ending:	2100									
Length of interval (a):	10									
Number of types for class RESID:	2									
Number of types for class EL:	4									
Number of types for class PH:	2									
Types of class RESID:	**Name**	**Lifetime (a)**	**flh (h/a)**	**Cooling-time (intervals)**						
	LWR-UOX-R	40	7884	1						
	LWR-MOX-R	40	7884	1						
Types of class EL	**Name**	**Lifetime (a)**	**flh (h/a)**	**Cooling-time (intervals)**						
	LWR-UOX	40	7884	1						
	LWR-MOX	40	7884	1						
	CAPRA	40	7884	1						
	ADS	40	7884	1						
Types of class PH:	**Name**	**Lifetime (a)**	**flh (h/a)**	**Cooling-time (intervals)**						
	HTR-UOX	40	7884	1						
	HTR-MOX	40	7884	1						
Capacities at the beginning of the first interval (GW-el):	**LWR-UOX-R**	**LWR-MOX-R**	**LWR-UOX**	**LWR-MOX**	**CAPRA**	**ADS**	**HTR-UOX**	**HTR-MOX**		
	111	9	0	0	0	0	0	0		
Total capacities (GW-el) at the end of intervals	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
class RESID	80	40	0	0	0	0	0	0	0	0
class EL	40	80	120	160	200	240	280	320	360	400
class PH	0	40	80	120	160	200	240	280	320	360

Closure for RESID at the end / new capacities at the beginning of intervals for EL and PH (GW-el)		1	2	3	4	5	6	7	8	9	10
RESID	LWR-UOX-R	37	37	37	0	0	0	0	0	0	0
	LWR-MOX-R	3	3	3	0	0	0	0	0	0	0
EL	LWR-UOX	40	37	37	37	75	71	69	66	91	95
	LWR-MOX	0	3	3	3	5	5	5	9	16	10
	CAPRA	0	0	0	0	0	3	3	2	0	0
	ADS	0	0	0	0	0	1	3	3	13	15
PH	HTR-UOX	0	40	40	37	37	75	75	75	75	115
	HTR-MOX	0	0	0	3	3	5	5	5	5	5

Table 1: Basic definition items of the scenarios are organised in an EXCEL-input file, called "vleem2-input.xls". The presented example is also content of the other Tables 2-4 shown below.

Comments to Table 1:

- The input-file starts with some basic definitions (time horizon, length of a "coarse" interval, in years)
- Next are the numbers of reactor types in the classes RESID, EL and PH.
- Each reactor type in these 3 classes is defined by name, lifetime (years), full load hours per year, and out-of-pile time ("cooling time") in numbers of "coarse" intervals.
- The RESID-reactors (with the post-fix "-R") have (by definition) existing capacities (GW-el) at the beginning of the time horizon. The EL-and PH-types have zero initial capacities (by definition). The RESID-reactors can die out only, there are no later capacity additions for them.
- For each of the 3 classes the time sequence of their total capacities (GW-el) is defined.
- For each RESID-reactor the number of retired capacities (in GW-el) per time interval is given ("closure"). It must correspond to the predefined total capacity dynamics and is checked by the programme. An error message is produced identifying the time step of the error. After correcting it the programme has to be restarted (see Chapter 3).
- For each EL- and PH-reactor type the newly built capacity is given for each time step under consideration of the retired capacities (defined by the lifetimes). It must correspond to the predefined total capacity dynamics and is checked by the programme. An error message is produced identifying the time step of the error. After correcting it the programme has to be restarted (see Chapter 3).

Column number: 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	U-nat-in	Th-in	FP-in	U-depl-in	U-irr-in	MA-in	Pu-f-in	Pu-rec-in	HM-in	Th/U-out	FP-out	U-depl-out	U-irr-out	MA-out	Pu-f-out	Pu-rec-out
Reload t/TWh																
LWR-UOX-R	18.7										0.1131	16.4	2.18	0.0023	0.0236	
LWR-MOX-R				1.6411			0.2132		1.854		0.1168		1.5725	0.013		0.152
LWR-UOX	18.7										0.1131	16.4	2.18	0.0023	0.0236	
LWR-MOX				1.6411			0.2132		1.854		0.1168		1.5725	0.013		0.152
CAPRA				0.05	0.2595			0.2475	0.557		0.1225		0.2598	0.015		0.1797
ADS					0.0056	0.0786		0.3953	0.4795		0.1192		0.0056	0.0656		0.2811
HTR-UOX	19.98										0.1041	18.72	1.1207		0.00156	
HTR-MOX	14.27	0.6382					0.1061		0.8143	0.606	0.1073	14.2	0.0112	0.0108		0.0329
First core t/GW																
LWR-UOX-R	491.44											431				
LWR-MOX-R				43.13			5.6		48.73							
LWR-UOX	491.44											431				
LWR-MOX				43.13			5.6		48.73							
CAPRA				1.59	8.254			7.873	17.717							
ADS					0.124	1.742		8.763	10.629							
HTR-UOX	242.67											214.42				
HTR-MOX	724.55	38.41					3.92		7.52			720.95				
Rest core t/GW																
LWR-UOX-R											2.142		85.209	0.0436	0.447	
LWR-MOX-R											2.782		76.676	0.31		8.714
LWR-UOX											2.142		85.209	0.0436	0.447	

LWR-MOX											**2.782**		**76.676**	**0.31**		**8.714**
CAPRA											**2.146**		**9.972**	**0.263**		**7.483**
ADS											**1.337**		**0.126**	**1.617**		**7.675**
HTR-UOX													**28.287**	**0.0125**	**0.259**	
HTR-MOX										**37.688**			**3.929**	**0.281**		**3.715**

Table 2: Data matrix for some reactor types ("Matrix.xls").
UOX = open cycle, MOX = mixed U-Pu-oxide, CAPRA = fast Pu-burner, ADS = accelerator-driven system,
HTR = high temperature gas-cooled reactor
Post-fix "-R": Symbolises RESID-reactors (already existing at the beginning of the time horizon). In the case presented here the two types of LWR-R reactors have the same specific mass flow data as the LWR-reactors newly built within the time horizon.

Storage at first period:						
Th/U-out	**FP-out**	**U-depl-out**	**U-irr-out**	**MA-out**	**Pu-f-out**	**Pu-rec-out**
10000	2500	500000	100000	60	600	600
Uirr-Waste	**Pu-Waste**	**MA-Waste**				
0	0	0				
Loss factors:						
VPU:	0.001					
VUIRR:	0.001					
VMA:	0.001					

Table 3: An example of "Storage.xls": Includes initial values of 7 storages and 3 waste components (details see text above) as tonnes of materials or of heavy metals (HM) and loss factors for reprocessing of Pu, irradiated U and minor actinides (MA). The waste-storages here are set to zero.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
LWR-UOX-R	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90
LWR-MOX-R	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90
LWR-UOX	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90
LWR-MOX	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90
CAPRA	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90
ADS	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90
HTR-UOX	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90
HTR-MOX	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90

Table 4: An example of "capf.xls" for n = 2: 10 coarse and 20 small time steps. All the capacity factors are here set to the value of 0.90 (7884 h/year), as in Table 1. But basically the value for each reactor and time step can be set to any value independent from the values taken for the "coarse" time steps where the capacity factors are only dependent from the reactor type, but constant over time.

- "vleem2-input.xls" and "capf.xls" as "txt"-files:
  Apply "xls_to_txt.exe" to these 2 files and save as ".txt".
  Results are "vleem2-input.txt" and "capf.txt".
- "Martrix.xls" and "Storage.xls" as "csv"-files:
  Apply "xls_to_txt.exe" to these 2 files and save as ".csv".
  Results are "Matrix.csv" and "Storage.csv".

Additionally required are the executable programmes

- "vleem2_mass.exe" (for the nuclear mass flow computations with the "coarse" time step), and
- "vleem2_use.exe" (for the nuclear mass flow computations with the "small" time step, but based on the nuclear reactor structure of the coarse time steps).

These two programmes are the "source code" of the nuclear mass balance model NMBM.

3. The execution of the programme

The batch-file "execute.bat" steers the execution of the NMBM.

Table 5 shows its structure (programme):

```
set topic_path=%CD%
cd %topic_path%
del error_output.txt
del error_output_use.txt
vleem2_mass.exe vleem2_input.txt Matrix.csv Storage.csv
if not exist error_output.txt vleem2_use.exe vleem2_use_input.txt capf.txt
```

Table 5: The structure of "**execute.bat**".

There you can change the names of the input-files, but then the actual input-files must renamed accordingly ! Attention: The name "*vleem2_use_input.txt*" cannot be changed, because this file is produced internally during the run of **execute.bat**!

To start the execution
double "click" on the file "execute.bat":
Initiates the execution of

- "vleem2_mass.exe" In a successful operation two output EXCEL-files are produced:
  - output_cap.xls: Contains the information on the dynamics of various capacities for the coarse time steps.
  - output_mass.xls: Details of the dynamics of the nuclear mass flows, of the storages and the waste-volumes for the coarse time steps.

  An exe-file *"create_charts.exe"* is automatically called from *vleem2_mass.exe* and creates Excel-diagrams from the result tables so that the user gets graphical pictures (3 diagrams of capacities and 4 diagrams of nuclear mass flows) of the results with the "coarse" time steps:
  Reactor capacities:
  7. Capacities divided according to the (three) reactor classes
  8. Capacities of all reactor types
  9. Capacity additions (new capacities) of all reactor types

Nuclear masses and mass flows:

9. Temporal storage contents of minor actinides, fresh and multiple recycled Plutonium (tonnes)
10. Consumption of natural Uranium (tonnes)
11. Consumption of fresh Thorium (tonnes)
12. Re-fabrication requirements (tonnes heavy metal/a)

The two files output_cap.xls and output_mass.xls are open and can be seen by pushing the icon "Microsoft Excel" in the task-bar. In this case you have to push the icon for **execute.bat** in the task-bar in order to proceed. If Excel is not open, then you can see the open file of **execute.bat**:
After successfully establishing these two files (they are open and must remain open at this intermediate stage of execution) **execute.bat** asks for the "number of short periods for JT": The number of "small" time steps within a "coarse" time step.
Put in this number and push the **enter** button.

If the structure of "capf.xls" corresponds precisely to this number **execute.bat** produces (without diagrams)

- output_use.xls: Details of the dynamics of the nuclear mass flows, of the storages and the waste-volumes for the small time steps.

- The quality of the results (for the coarse time steps) can be examined easily by controlling the diagrams in output_cap.xls and output_mass.xls:
  - If the results are **not satisfactory**, then close the two files output_cap.xls and output_mass.xls **without saving them.**
  - If the results are **satisfactory** and should be stored, then **save these two files under other names together with a change of the file-type from "*.txt" to "*.xls".**
- In the error case an error file "error_output.txt" or "error_output_use.txt" is produced, indicating the type of error. In such a case the corresponding EXCEL input-xls-file has to be opened, the correction be made, closed and saved (by applying "xls_to_txt.exe") as ".txt"- or ".csv"-file, respectively. "**execute.bat**" can then be re-started with a double-click.
- **Attention**:
  - Changes in an **input file** have always to be **saved!**
  - Before (re)starting "**execute.bat**" all output-files (output_cap.xls, output_mass.xls and output_use.xls) must be closed (otherwise they could not be overwritten with new results), **but not saved**.
  - The names of the output-files (output_cap.xls, output_mass.xls and output_use.xls) cannot be changed. For use under other names they have to be copied first and then renamed.
- If input-files with other names should be used, then they must be in the same file-domain as the batch-file "**execute.bat**", and in this batch-file the old names must be changed to the new names of the input-files **by applying the right mouse-button to the file "execute.bat" and then pushing "process" (German: "Bearbeiten")** (the next option behind "open"):
  ► See **Table 5**:
  - Line 5: If the files quoted above as "vleem2_input.xls" and/or "Matrix.xls" and/or "Storage.xls" get other names, then the corresponding ".txt"- and ".csv"-files in "**execute.bat**" must get the same names as these ".xls"-files.
  - Line 6: If the file quoted above as "capf.xls" gets another name, then "capf.txt" must get the same new name in "**execute.bat**".

- All other file-names must be kept in the original version:
  vleem2_mass.exe
  error_output.txt
  vleem2_use.exe
  vleem2_use_input.txt
- After having changed the file "**execute.bat**" press **"save"** before the next run of "**execute.bat**".

4. An important consistency check

The rationale of the main direction in long-term reactor development is the destruction of stockpiles of

- Plutonium (Pu),
- minor actinides (MA),
- some long-lived fission products (FP)
  (e.g., I-129, Cs-135, Tc-99, Sn-126, Se-79; realistic destruction capabilities, if at all, only for Iodine I-129 and Technetium Tc-99).

Table 2 (matrix of specific mass flow data for the considered reactors) shows that it contains also the specific charges and discharges of fresh Pu (Pu-f), multiple recycled Pu (Pu-rec), MA and FP (as a whole):

- Inputs: Columns 3 ,6, 7, 8
- Outputs: Columns 11, 14, 15, 16

Table 3 defines the storages (with initial values) for the FPs, MA, fresh Puu (Pu-f) and for the total Pu (Pu-rec, fresh + multiple recycled).
Currently there are no data for reactors destroying long-lived FPs by loading them from FP-storages, but Table 2 comprises reactors burning fresh and/or recycled Pu: All MOX-reactors, CAPRA and ADS. Only ADS is capable here of MA-destruction.

The main consistency checks are twofold:

6. The Pu-stockpiles must not be negative.
7. The total Pu-stockpile (Pu-rec) must not be smaller than the Pu-f stockpile.

If the Pu-stockpile results (last page of the EXCEL-file "output_cap.xls) of a run of "executive.bat" contradict these checks, then the capacities of newly built reactor types have to be changed accordingly in "vleem2-input.xls" (see Table 1 for the required consistency of corrections: if one capacity addition is raised, then another must be reduced by the same amount) in the lines below "Closure / new capacity in interval" in the periods before and/or in the same period where this contradiction appears the first time.
Prior to the new run of "execute.bat" do not forget to **save** and **close** the corrected "vleem2_input.xls", apply "xls_to_txt.exe" to this file and save it as "vleem2_input.txt"; and close "output_mass.xls", if it is still open.

The input data presented in the Tables 1 to 4 above (for demonstrative purposes, although being more or less realistic for a "High Nuclear Case" for western Europe) produce "reason-able" results with regard to the two consistency checks quoted above, as shown below in Figure 1. Figures 2 and 3 show the corresponding capacity distributions and capacity additions. Time horizon is 2000-2100, with the "coarse" time steps of 10 years, i.e. 10 periods (see Table 1). The "small" time steps taken here are 5 years, i.e. 20 periods (see Table 4), but their results are not presented here. They are in accordance with the results of the coarse time steps (as can be easily checked by running "execute.bat" with the same data as above).

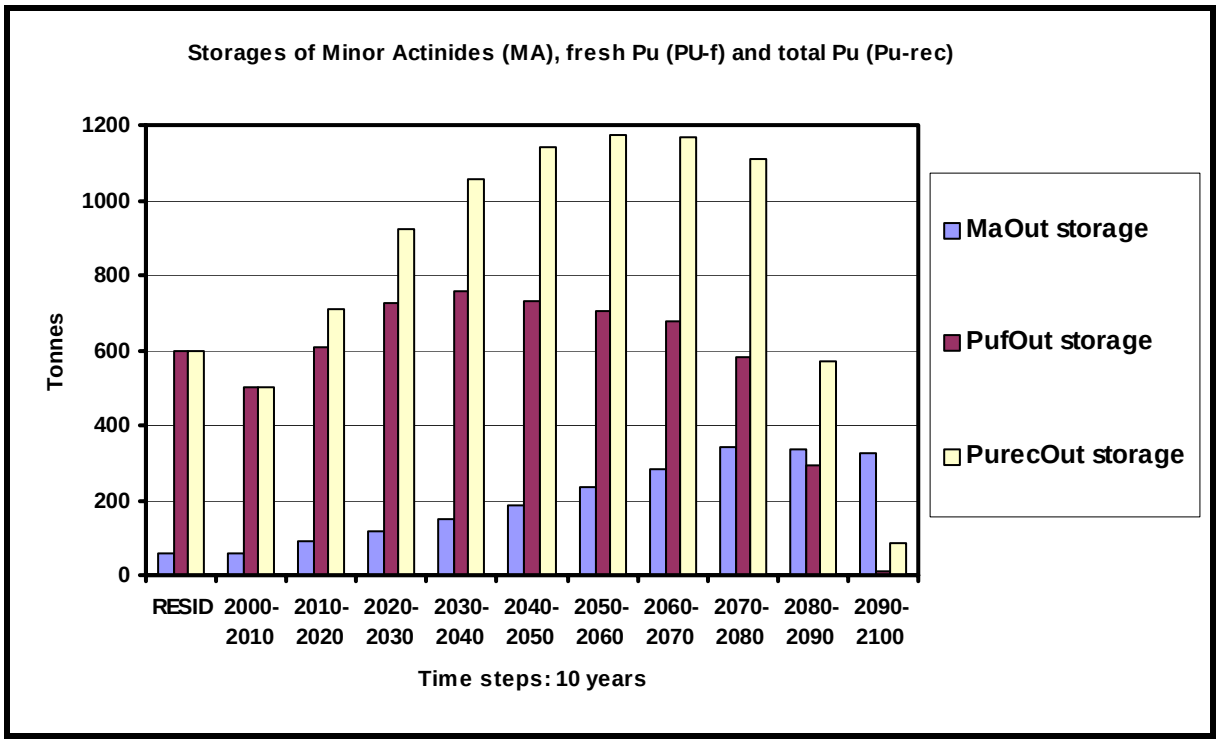


Figure 1: Dynamics of the storages for minor actinides – MA (blue), fresh (and once recycled) Pu – Puf (red), and total Pu – Purec (yellow).

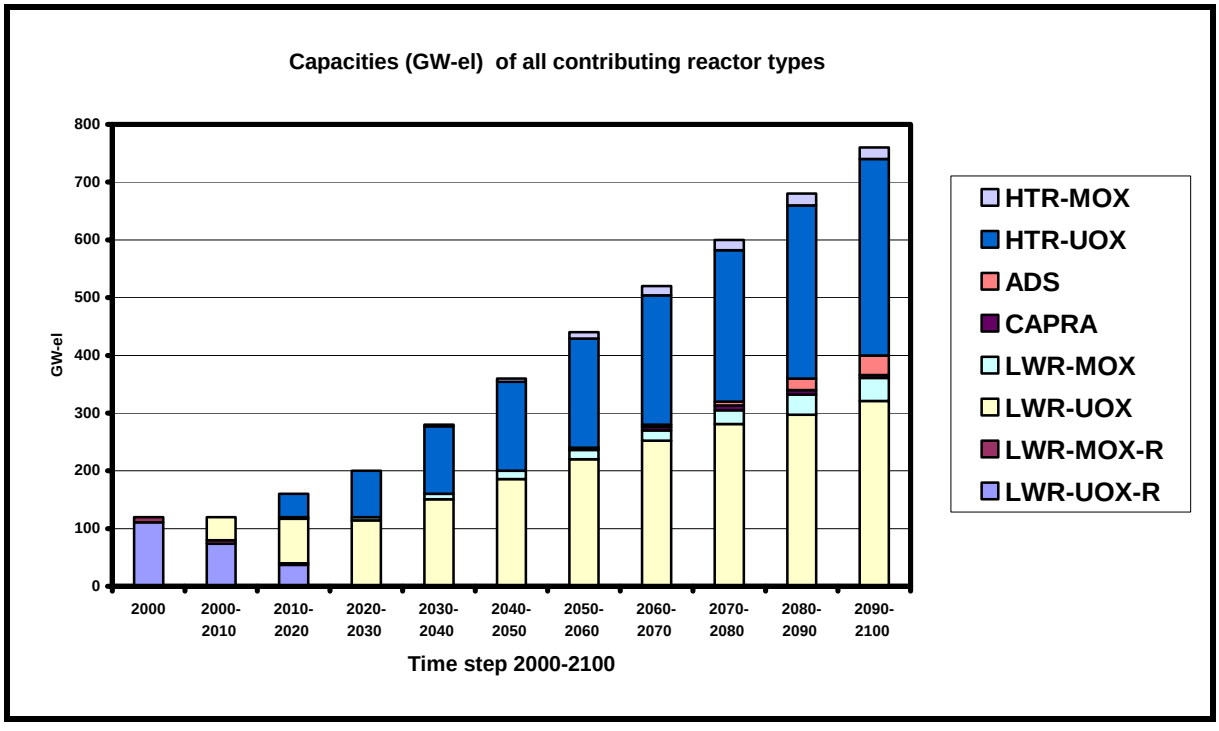


Figure 2: Dynamics of the capacities (GW-el) of the 9 contributing reactor types. Observe that the two RESID types (marked by "-R") can only die out, but not be added again.

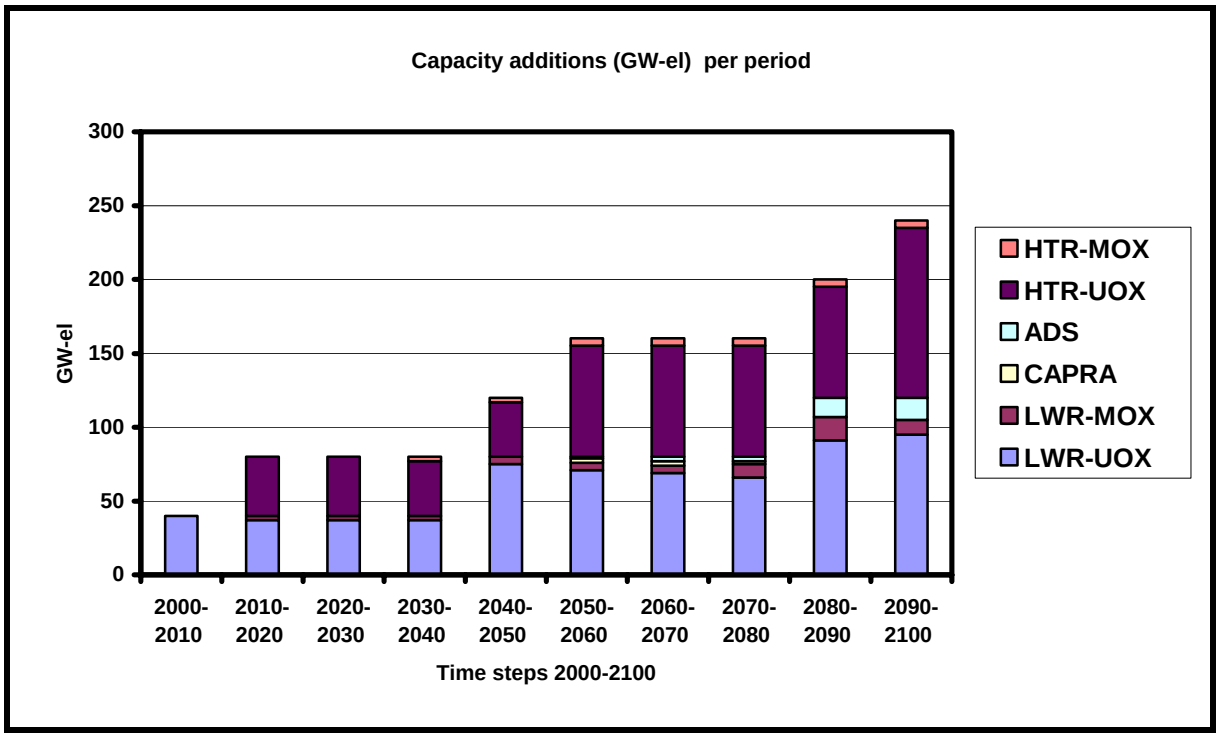


Figure 3: Dynamics of the capacity additions (GW-el per period) of the 7 contributing reactor types beyond the initially existing RESID-reactors. Here the specific data for the LWRs are independent from the membership to a reactor class.

Apparently it seems to be possible to reduce Pu-stockpiles even in cases of a pronounced nuclear expansion by extended additions of Pu-destroying reactors, but the reactor types included here do not reduce distinctly the MA stockpile. CAPRA and ADS are implemented first in the 6th period (2050-2060). It is not expected that these types can be commercially available earlier, due to the complex development steps for the reactors and their fuel cycles.

In Figures 2 and 3 it can be seen that the open-cycle types LWR-UOX and HTR-UOX remain dominating. The required fuel supply needs fresh U (and to a minor extent fresh Th). The reactor pool defined here cannot live alone from recycled fuel. There are two main reasons:

3. The capacity expansion is very pronounced.
4. No fast breeders are applied, only a fast reactor (CAPRA) as Pu-burner. A fast breeder pool could live quite well a long time from the stockpiles of depleted U (tails from enrichment plants). The lack of breeder reactors leads in this demonstration case also to a high consumption of natural U, as Figure 4 shows (next page):

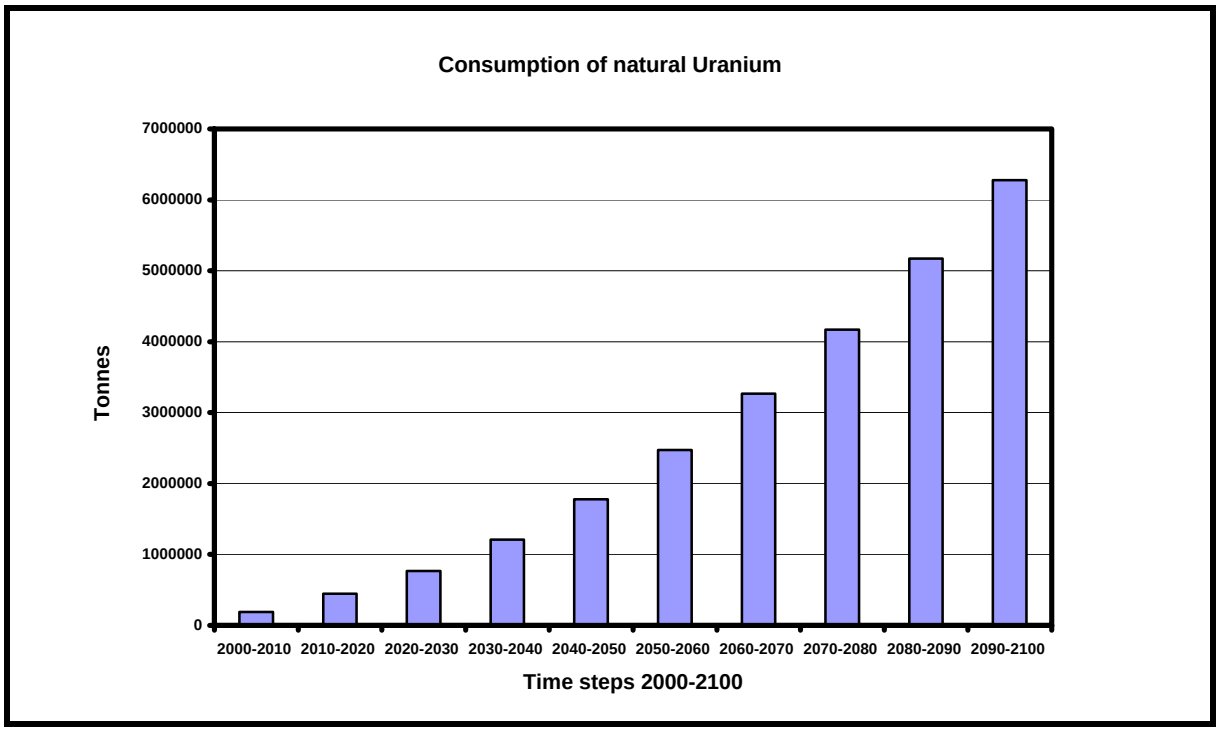


Figure 4: Dynamics of the natural-U consumption of the demonstration strategy. This case apparently leads to a very high U-consumption, nearly twice that much as there are known mineral U-reserves with production costs below 130 US$/kg U (~ 4 million tonnes).

Annex

The Nuclear Mass Balance Model (NMBM)

1. Introduction

The model consists of 3 parts (modules):

- The capacity programme
- The mass balance programme
- A programme to perform computations with deliberately small time steps based on the results of (more or less) coarse time steps of a size of (normally) some years, e.g., 5 or 10 years within a time horizon of some decades, say a 100 years.
  This 3rd module is mainly of interest,
  1. if the model is part of a comprehensive (superior) energy model with nuclear contributions, and
  2. if this energy model simulates the operation of a plant park in small time steps (months, weeks, days or less).

The NMBM presented below comprises therefore 3 time loops:

7. In the first loop the "coarse" time sequence of individual reactor capacities are calculated via the assumptions of individual capacity additions.
8. In the second loop the corresponding mass balances are calculated, considering operating capacities, capacity additions and retirements.
9. In the third loop the mass balances are calculated with deliberately small (constant) time steps and deliberately chosen load factors for each reactor type and time step (independent from the load factors in the "coarse" calculation where these factors are only reactor-dependent, not variable with time), but based on the dynamic reactor pool as defined for the "coarse" time steps (first and second loops).

2. Inputs for the capacity programme

2.1 Generic input parameters

Time horizon and loss factors of reprocessing:

First and last year of the time horizon (calendar years):	JI, JF	(e.g., 2000, 2100)
Time horizon (years):	JTH: JTH = JF – JI	(e.g., JTH = 100 years)
Length of time steps (years):	JT, divisor of JF – JI = JTH	(e.g., JT = 10 years)
Number of time steps:	IT = JTH / JT	(e.g., IT = 10)

Loss factors for the reprocessing of irradiated Uranium (U), Plutonium (Pu), and Minor Actinides (MA): VUIRR, VPU, VMA
Typically these loss factors are between 0.01 and 0.001.

In order to comply with the implemented formulas (see later) these values are corrected to
VUIRR = VUIRR/(1 + VUIRR) VPU = VPU/(1 + VPU) VMA = VMA/(1 + VMA)

It should be pointed out that the explanations below are mainly done or supported by the use of mathematical notation and FORTRAN-like statements for the sake of better understanding.

2.3 Nuclear capacities

2.2.1 Classes:

The capacities are divided into 3 "classes":

10. RESID: Nuclear plants (NP) already existing at the begin of the time horizon
11. EL: Electricity producers
12. PH: Process heat producers

Total capacity = RESID + EL + PH (GW-el)

All classes are expressed in electrical units: Gigawatt-electric – GW-el
Therefore the capacities are additive !

2.2.2 Definition of the total capacities for RESID, EL, PH

Prescribed for each time step I from I = 1 up to I = IT:
TCAP(Class, I): TCAP(RESID, I), TCAP(EL, I), TCAP(PH, I)

ToCAP(I) = TCAP(RESID, I) + TCAP(EL, I) + TCAP(PH, I) **total capacity**

2.2.3 Definition of reactor types

Separated according to the classes RESID; EL, PH: Type (Class)

Identification of the reactor types by their acronyms (LWR, HTR, etc. and further additions, like LWR1, LWR2, LWR-UOX, LWR-MOX, HTRold, HTRnew, HTR-PB /PB = pebble bed/).

Identification of the reactor types:

- Acronym of the name
- Plant life time: LT(Type(Class)): Years, must be a multiple of the time step JT
- Capacity factor: Full-load hours per year (year = 8760 h): FLH(Type(Class))
- Cooling times of the discharged fuel: ICT(Type(Class)): A multiple of the time step JT: 1, 2, 3, ….

Class RESID of the initially existing NPs:

- Number of reactor types and their names (acronyms)
- For each RESID-reactor type the decreasing capacity for each time step (until the capacity is – and remains – zero.
- The assumptions must fit into the prescribed time sequence of the total RESID-capacity !

Classes EL and PH as new NP-types:

- Number of reactor types and their names (acronyms)
- Time sequence of the total capacities for EL and PW: TCAP(Class, I)
- Time sequence of new plant additions for each Type(Class):
  CAPnew (Class, Type(Class); I) for each time step I
- **Control:**

$$TCAP_{Test}(Class, I) = TCAP(Class, I-1) + \Sigma\, CAPnew\,(Class, Type(Class), I) -$$

$$- \sum_{(Type(Class))} CAPnew\ (Class, Type(Class), I - LT(Type(Class)))$$

Diff (Class, I) = $TCAP_{Test}$(Class, I) – TCAP(Class, I)

Diff (Class, I) must be zero !

If DIFF (Class, I) > 0: Reduce CAPnew (Class, Type(Class), I)

If DIFF (Class, I) < 0: Enlarge CAPnew (Class, Type(Class), I)

Important: There are CAPnew-data only for the classes EL and PH !
The RESID-class is dying out !

2.2.4 Calculation of the individual capacities

If finally for all time steps from I = 1 to I = IT: DIFF(Class, I) = 0,
then the individual nuclear capacities can be calculated:

CAP(Class, Type(Class), I) =
= CAP(Class, Type(Class), I-1) + CAPnew(Class, Type(Class), I) –
– CAPnew(Class, Type(Class), I-LT(Type(Class)))

Retirements:
For RESID:
RET(RESID, Type(RESID), I) = CAP(RESID, Type(RESID), I-1) – CAP(RESID, Type(RESID), I)
For EL and PH:
RET(Class, Type(Class), I) = CAPnew(Class, Type(Class), I-LT(Type(Class)))

Electricity- and process heat production of each plant during time step I:
PRO = "product", expressed in electrical Terawatt-hours (TWH-el)

PRO(Class, Type(Class), I) = CAP(Class, Type(Class), I)∗FLH(Type(Class))∗JT (single plant)

$$CPRO(Class, I) = \sum_{(Type(Class))} PRO(Class, Type(Class), I)$$ all plants of a Class

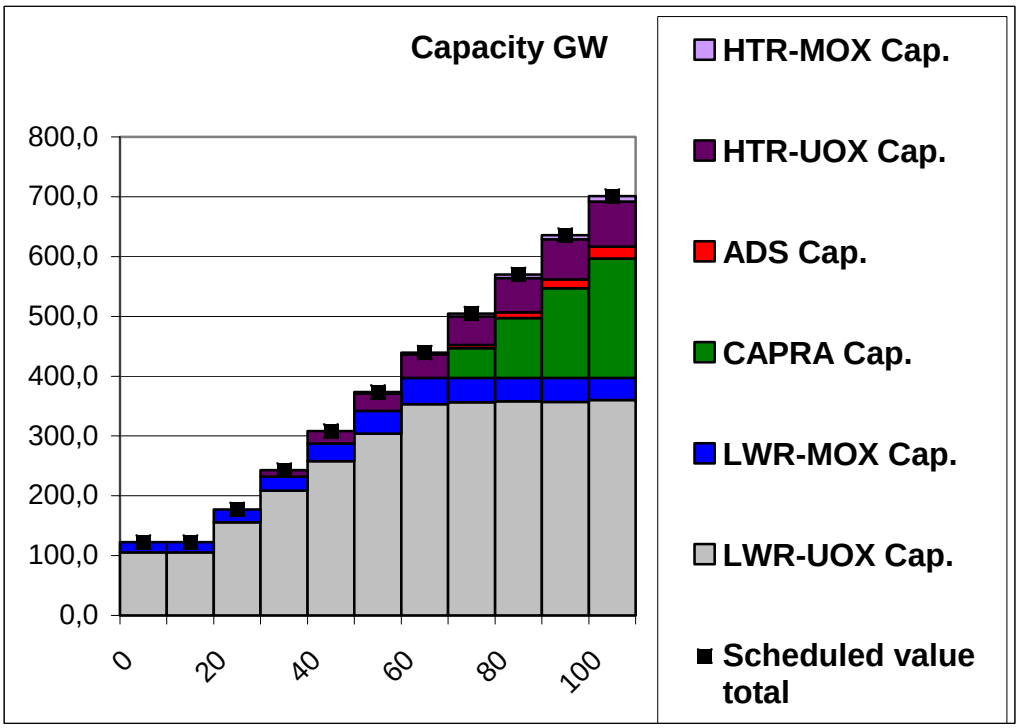
$$ToPRO(I) = \sum_{Class} CPRO(Class, I)$$ all plants

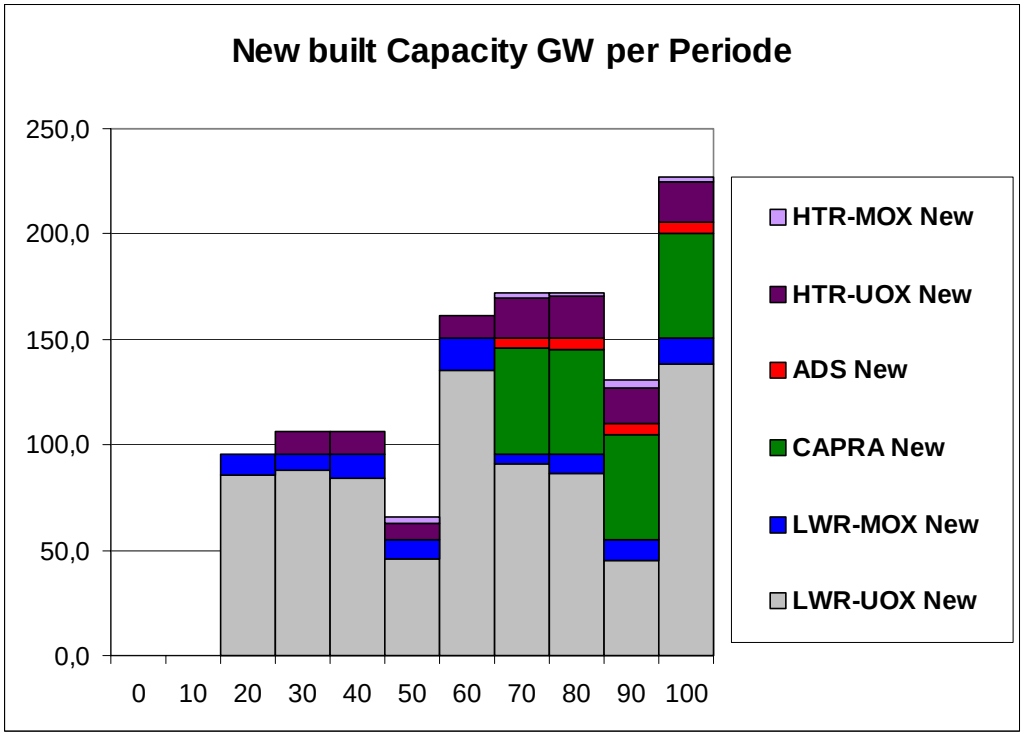
The computations are done for all time steps from I = 1 to I = IT:
Results are time series of all

- capacities,
- new installations,
- expirations,
- electricity and process heat productions (expressed in TWh_{el}).

These results allow the computation of the nuclear mass flows.

The diagrams on this page show for the sake of clarity the capacity results as derived with the former EXCEL-programme of VLEEM1:
Inputs were the total capacities for electricity and process heat (the latter for the HTRs, expressed as GW_{el}), and then the capacity additions for each reactor type to fill up the prescribed total capacities.





3. Computations of the nuclear mass flows

3.1 The nuclear fuel cycle

The nuclear fuel cycle is divided into 5 steps (Figure 3.1):

- Fresh heavy metals (HM): Natural Uranium – U_{nat}, and Thorium – Th
- Fuel element-fabrication - fresh and recycled ones ("re-fabrication")
- Reactor operation
- Interim storage for spent fuel elements, then possibly reprocessing and re-fabrication
- Final repository

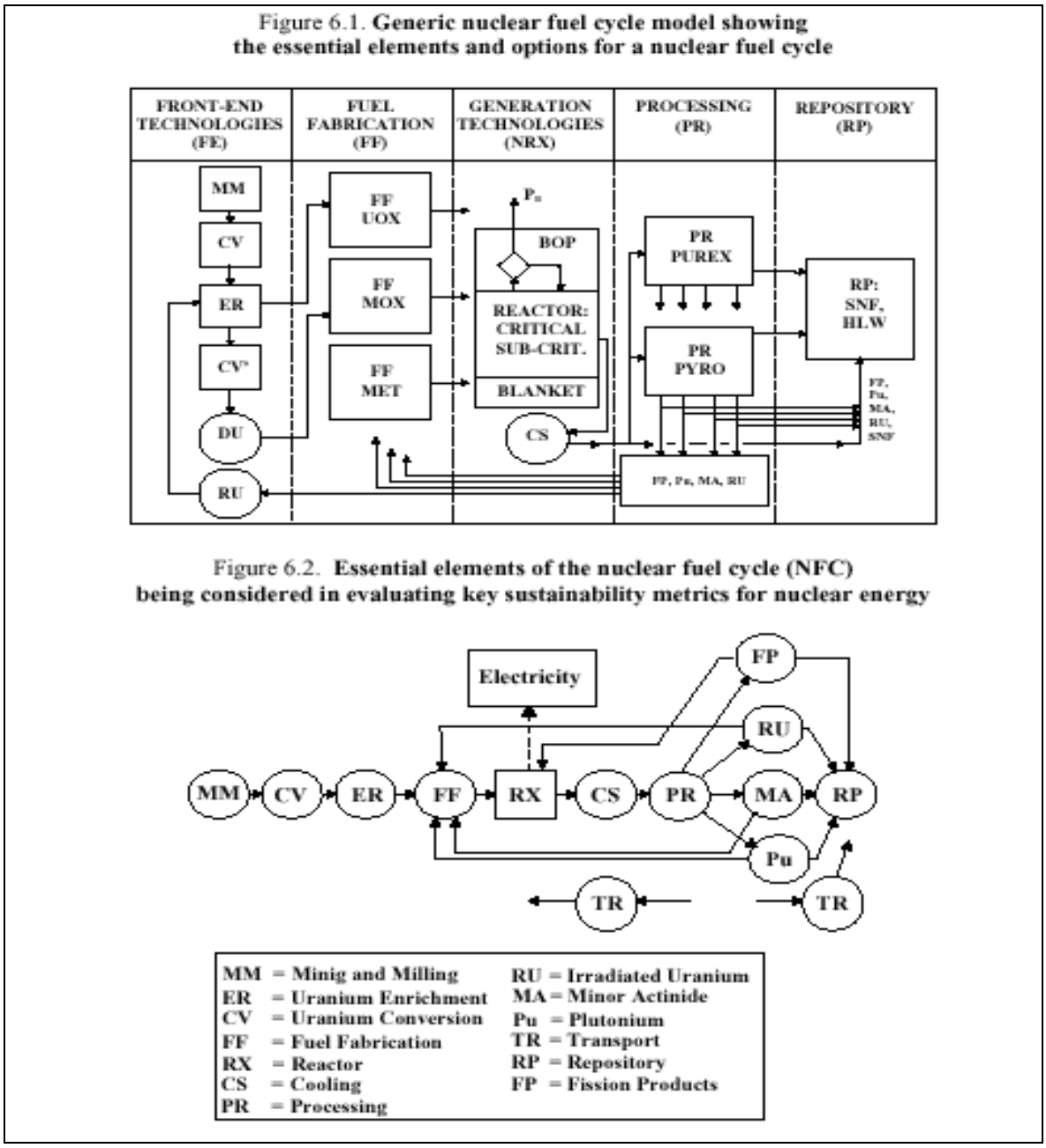


Figure 3.1: Generic nuclear fuel cycle model (upper part) and the essential elements of the nuclear fuel cycle (lower part): DU = depleted U, MET = metal fuel /NEA 2002/

The nuclear fuel cycle supplies not only one reactor or one reactor type, but an entire, more or less coupled system of reactor types, schematically shown in Figure 3.2:

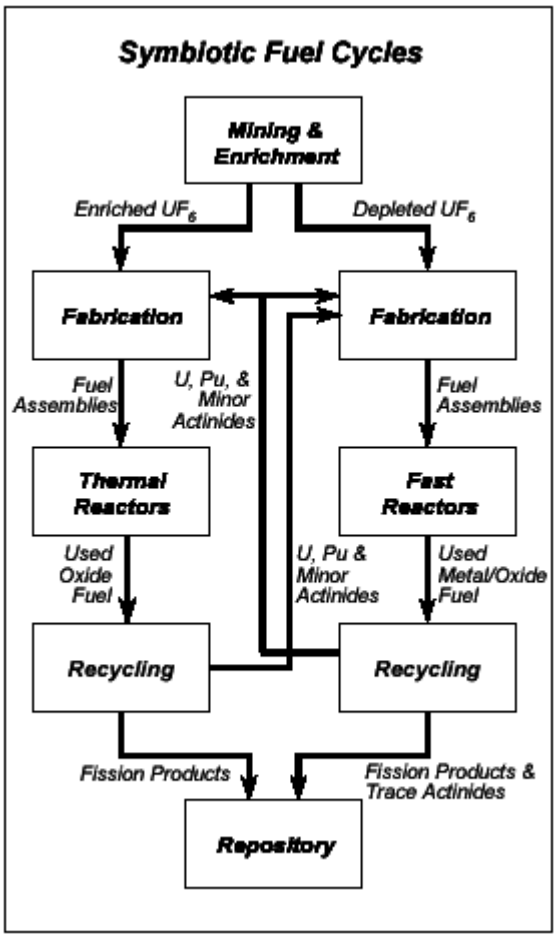


Figure 3.2: Scheme of a symbiotic fuel cycle comprising the interference between thermal and fast reactors (recycle of actinides from thermal into fast reactors) /NERAC 2002/

Each of the reactors discharges into each interim storage (possibly also zero) and can take material from each storage (via re-fabrication, possibly also zero).
The numerical description goes via 8 "pots" and the final disposal:
Fresh fuel requirements **(per time step and cumulated):**

- U_{nat}
- Th

Interim storages:

- Fission products (FP),
- Depleted U (U-depl),
- Irradiated U (U-irr),
- Minor actinides (MA)
- Fresh (once recycled) Pu (Pu-f)
- Multiple recycled Pu (Pu-rec)

Final disposal:

- Irradiated (spent) Th and U destined for repository (Th/U-out)
- Waste: Pu and MA (entire discharges or only their re-fabrication losses), FP, U, Th

For the interim storages initial values ("RESIDs") must be set (see Chapter 3.4 below).

In the case of fresh fuel the inputs require the amount of fresh natural material: U_{nat} and Th. The enrichment-value is not used explicitly in this model (see Chapter 3.2).

Fuel fabrication:

- Fresh fuel (U_{nat} and Th)
- MOX-re-fabrication: Mixed-oxide fuel made of U and/or Th and Pu
- Re-fabrication for fast reactors (FR): Material from all interim storages, possibly also fresh HM
- ADS-re-fabrication: Material from all interim storages, possibly also fresh HM

3.2 Description of the individual reactor types

Table 3.1 shows the description matrix for the reactors contributing to a scenario computation, here defining 3 classes, each with 2 reactor types. These classes are only of relevance for the computation of capacities (operating, new and retiring ones). For the mass flow computations this reference to classes is skipped, because here being irrelevant (see below).

Each reactor type has 3 data blocks:

- First core (t/GW-el)
- Rest core (t/GW-el)
- "Reloads" (charges) and discharges ($T/TW_{el}h$)

Each reactor has 16 components for these 3 items:

- Columns 1 – 9 for charges,
- Columns 10 – 16 for discharges.

Matrix elements without a number are set to zero.

Charges (columns 1 to 9, called "Reload") ($t/TW_{el}h$):

10. U-nat-in: Natural U calculated according to the required enrichment of fresh U-fuel.
11. Th-in: Charge of fresh Th.
12. FP-in: Charge with fission products (FP)
13. U-depl-in: Charge with depleted U
14. U-irr-in: Charge with recycled U
15. MA-in: Charge with minor actinides (MA)
16. Pu-f-in: Charge with "fresh" (only once reprocessed) Pu
17. Pu-rec-in: Charge with multiple recycled Pu
18. HM-in: Charge with heavy metal containing reprocessed material (for the computation of re-fabrication requirements)

Discharges (columns 10 to 16 of "Reload") ($t/TW_{el}h$):

10. Th/Pu-out: Discharge of irradiated Th and/or U for final disposal
11. FP-out: Discharge of fission products (FP)
12. U-depl-out: Tails of fresh enriched U ("depleted" U), "waste" of U235-enrichment
13. U-irr-out: Discharge of irradiated U
14. MA-out: Discharge of MA
15. Pu-f-out: Discharge of "fresh" Pu (originating from fresh U)
16. Pu-rec-out: Discharge of recycled Pu

Table 3.2 shows a data matrix of reactor types with data (except zeros in empty spaces), without reference to classes.

Column number:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	U-nat-in	Th-in	FP-in	U-depl-in	U-irr-in	MA-in	Pu-f-in	Pu-rec-in	HM-in	Th/U-out	FP-out	U-depl-out	U-irr-out	MA-out	Pu-f-out	Pu-rec-out
Reload t/TWh																
Class 1: NPP-Type 1																
NPP-Type 2																
Class 2: NPP-Type 3																
NPP-Type 4																
Class 3: NPP-Type 5																
NPP-Type 6																
First core t/GW																
Class 1: NPP-Type 1																
NPP-Type 2																
Class 2: NPP-Type 3																
NPP-Type 4																
Class 3: NPP-Type 5																
NPP-Type 6																
Rest core t/GW																
Class 1: NPP-Type 1																
NPP-Type 2																
Class 2: NPP-Type 3																
NPP-Type 4																
Class 3: NPP-Type 5																
NPP-Type 6																

Table 3.1: Schematic data matrix for 6 reactor types with formal identification of their relationships to the 3 classes.
NPP = nuclear power plant.
There must be a number in each matrix element, or zero, because sums are running along rows and columns.

Column number:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	U-nat-in	Th-in	FP-in	U-depl-in	U-irr-in	MA-in	Pu-f-in	Pu-rec-in	HM-in	Th/U-out	FP-out	U-depl-out	U-irr-out	MA-out	Pu-f-out	Pu-rec-out
Reload t/TWh																
LWR-OC	20.513										0.123	18.119	2.2385	0.0036	0.0288	
LWR-MOX				2.204			0.194		2.398		0.122		2.128	0.018		0.1322
CAPRA-MOX				0.050	0.2595			0.2475	0.557		0.1225		0.2598	0.015		0.1797
TRU-burner				0.0543	0.4688	0.0307	0.0489	0.1788	0.7815		0.1088		0.4692	0.0245		0.179
FR (CR = 1)	0.1128				1.9328	0.0056		0.1816	2.2328		0.1124		1.933	0.0056		0.1818
TRU-ADS					0.0056	0.0786		0.3953	0.4795		0.1192		0.0056	0.0656		0.2811
First core t/GWe																
LWR-OC	539.10											476.20				
LWR-MOX				57.90			5.10		63.0							
CAPRA-MOX				1.590	8.254			7.873	17.717							
TRU-burner				2.032	17.547	1.149	1.830	6.693	29.251							
FR (CR = 1)	4.231				72.491	0.210		6.811	83.743							
TRU-ADS					0.124	1.742		8.763	10.629							
Rest core t/GWe																
LWR-OC											2.04		76.87	0.06	0.478	
LWR-MOX											2.02		71.78	0.298		5.405
CAPRA-MOX											2.146		9.972	0.263		7.483
TRU-burner											2.188		19.959	1.110		8.180
FR (CR = 1)											0.763		27.007	0.076		2.467
TRU-ADS											1.337		0.126	1.617		7.675

Table 3.2: Data matrix for some reactor types. OC = open cycle, MOX = mixed U-Pu-oxide
TRU = excess transuranics from fast breeders, FR = fast reactor, CR = conversion ratio (of U238 to Pu)
CAPRA = fast Pu-burner, ADS = accelerator-driven system

Definition: **Types ≡ number of all reactor types**

The data structure is of the mode
MATRIX(rows, columns) = MATRIX(3 x number of all reactor types, MAT) =
= MATRIX(3 x Types, MAT)
Rows of MATRIX from 1 to 3 x Types:

Type = 1 to Types:	Rows related to "Reload"
Type = Types+1 to 2 x Types:	Rows related to "First core"
Type = 2 x Types +1 to 3 x Types:	Rows related to "Rest core"

Columns of MATRIX from MAT = 1 to MAT = 16:

MAT = 1 to 9:	Specific material inputs (charges) into the reactors
MAT = 10 to 16:	Specific material outputs (discharges) of the reactors

3.3 Formulas for the specific reactor charges

The reactor types are now numbered from 1 to "Types" without further reference to classes !

Specific values MATIN(MAT; I) for the nuclear substances (materials) MAT = 1 to 9 for the time step I = 1 to IT:

The formulas are written in a FORTRAN-similar notation, but the model is written in C++ !

The basic formula for re-fabrication sums only over reactors "Refab-Type" using re-fabricated material (CAPnew is the capacity (GWe) of reactor type "Refab-Type" newly installed per time step I, CAP is the capacity of reactors type "Refab-Type" (GWe) operated within the period I under consideration):

Refab (t HM/a, I) =

$$= \{ \sum_{\text{Re } fab-Type} \textbf{CAPnew(Refab-Type, I) x HM-loading}_{\textbf{first core}}\textbf{(Refab-Type)}\}/JT +$$

$$+ \sum_{\text{Re } fab-Type} \textbf{CAP(Refab-Type, I) x HM-loading(Refab-Type) per annum } \}$$

MAT =1:9 **Inputs into the individual reactor types:**

$$MATIN(MAT,I) = \sum_{Type=1}^{Types} PRO(Type, I) * MATRIX(Type, MAT) +$$

$$+ \sum_{Type=1}^{Types} CAPnew(Type, I) * MATRIX(Types+Type, MAT)$$

```
Do I = 1 to IT
 Do MAT = 1 to 9
  MATIN(MAT, I) = 0
  Do Type = 1 to Types
   TMATIN1(Type, MAT, I) = PRO(Type, I) * MATRIX(Type, MAT)
   TMATIN2(Type, MAT, I) = CAPnew(Type, I) * MATRIX(Types + Type, MAT)
   MATIN(MAT, I) = MATIN(MAT, I) + TMATIN1(Type, MAT, I) + TMATIN2(Type,
   MAT, I)
  end Type
 end MAT
```

```
 Refab(I) = MATIN(9, I)/JT          Annual re-fabrication amounts (t HM/a) in time
                                    step I
 UNAT(I) = MATIN(1, I)              Unat-requirements in period I (t/JT)
end I
UNATT(1) = UNAT(1)
THT(1) = MATIN(2, 1)

Do I = 2 to IT
 UNATT(I) = UNATT(I-1) + UNAT(I),   Cumulated Unat-requirements up to period I
                                    (tonnes t)
 THT(I) = THT(I-1) + MATIN(2, I),   Cumulated Th-requirements up to period I
                                    (tonnes t)
end I
```

3.4 Formulas for the specific reactor discharges

Materials in storages:
Formulas for STORAGE(MAT, I) and TMATOUT(MAT, I) for the substances MAT = 10 to 16 in the time steps I = 1, 2, ... ,IT

MAT = 10:16 **Material outputs from the individual reactor types:**

```
Do I = 1 to IT
 Do MAT = 10 to 16
  Do Type = 1 to Types
   TMATOUT1(Type, MAT, I) = PRO(Type, I) ∗ MATRIX(Type, MAT)
   TMATOUT2(Type, MAT, I) = Still(Type, I) ∗ MATRIX(2∗Types + Type, MAT)
   TMATOUT(Type, MAT, I)=TMATOUT1(Type, MAT, I) + TMATOUT2(Type, MAT, I)
  end Type
 end MAT
end I
```

Balance of the storages for FPs, U-depl, U-irr, MA, PU-f and total Pu (MAT = 10 to 16):

STORAGE(MAT, 1) = RESID-values (initial values) for MAT = 10 to 16

STORAGE(16, I) = STORAGE(Pu-total: Fresh or once recycled plus multiple recycled Pu, I)

```
Do I = 1 to IT
 MAT = 10:15 !!
 Do MAT = 10 to 15
  FACTOR = 0
  Do Type = 1 to Types
   If (I-ICT(Type)) > 0 then FACTOR = FACTOR + TMATOUT(Type, MAT, I-ICT(Type))
  end Type
  if I = 1 then STORAGE(MAT, 1) = STORAGE(MAT, 1) + FACTOR
  else STORAGE(MAT, I) = STORAGE(MAT, I-1) + FACTOR
  If MAT > 10 then STORAGE(MAT, I) = STORAGE(MAT, I) – MATIN(MAT-8, I)
 end MAT
```

```
 MAT = 16:
 if I = 1 then STORAGE(16, 1) = STORAGE(16, 1) + FACTOR – MATIN(7, 1)
 else STORAGE(16, I) = STORAGE(16, I-1) + FACTOR – MATIN(7, I)
 FACTOR = 0
  Do Type = 1 to Types
   If (I-ICT(Type)) > 0 then FACTOR = FACTOR + TMATOUT(Type, 16, I-ICT(Type))
  end Type
 STORAGE(16, I) = STORAGE(16, I) + FACTOR – MATIN(8, I)
end I
```

3.5 Computation of the waste-amounts

The waste comprises the cumulated discharged (spent) amounts of

- Th, FPs, non-re-fabricated amounts of U, MA, and PU, and
- re-fabrication losses of losses of U, Pu, and MA

```
UIRRwaste(1) = RESID-value
PUwaste(1) = RESID-value
MAwaste(1) = RESID-value

VPU, VUIRR, VMA:  Loss factors at reprocessing of U-irr, Pu and MA (see first page)

Do I = 1 to IT
 If I = 1 then do
  UIRRwaste(1) = UIRRwaste(1) + VUIRR∗MATIN(5, 1)
  PUwaste(1) = PUwaste(1) + VPU∗(MATIN(7, 1) + MATIN(8, 1))
  MAwaste(1) = MAwaste(1) + VMA∗MATIN(6, 1)
 end if
 else do
  UIRRwaste(I) = UIRRwaste(I-1) + VUIRR∗MATIN(5, I)
  PUwaste(I) = PUwaste(I-1) + VPU∗(MATIN(7, I) + MATIN(8, I))
  MAwaste(I) = MAwaste(I-1) + VMA∗MATIN(6, I)
 end else
 WASTE(I) = STORAGE(10, I) + STORAGE(11, I) + UIRRwaste(I) + PUwaste(I) +
          + MAwaste(I)
end I
```

List of the required RESID-values (initial values at the beginning of the first period):

```
STORAGE(MAT, 1) = RESID-values for MAT = 10 to 16
UIRRwaste(1) = RESID-value
DPUwaste(1) = RESID-value
DMAwaste(1) = RESID-value
```

4. Computations with deliberately small time steps

4.1 Generic aspects

These computations are based on the reactor structure as defined above with the “coarse” time step, but proceed along small constant time steps and freely fixable capacity (load) factors,

separately for each reactor type and time step. Above, for the computations with the coarse time steps, the capacity factors were only reactor type dependent, but constant in time: FLH(Type).

These "use"-computations run with deliberately short constant time steps, as integer fraction of the coarse step JT:

⇒ IN Number of short time steps in JT
⇒ N = IT∗IN Total number of the small time steps:
J = 1 to N

BT = JT/IN length of the small time step (a), e.g., ¼ [year]
Time unit is [year].

The load CAPF(Type, J) = capacity factor of the reactor „Type" in the small time step J, is an input data in this module, but also can be transferred from external programmes:
CAPF(Type = 1 to Types, J =1 to N) [0 ≤ CAPF ≤ 1

4.2 Computation of the mass balances for the individual reactor types

Definition of matrix dimensions:
DSTORAGE(MAT = 10 to 16, J = 1 to N)
DMATIN(MAT= 1 to 9, J = 1 to N)
DUNAT(J = 1 to N)
DUNATT(J = 1 to N)
DTHT(J = 1 to N)
DPUwaste(J = 1 to N), DMAwaste(J = 1 to N), DUIRRwaste(J = 1 to N),
DWASTE(J = 1 to N)

```
TMATIN1 = TMATIN1/IN
TMATIN2 = TMATIN2/IN
TMATOUT1 = TMATOUT1/IN
TMATOUT2 = TMATOUT2/IN

DUNATT(1) = 0
DTHT(1) = 0

Do J = 1 to N
 Do Type = 1 to Types
  CAPF(Type, J) = CAPF(Type, J)*8.76/FLH(Type)
 end type
I = 1 + (J-1)/IN
 Do MAT = 1 to 9
  DMATIN(MAT, J) = 0
  Do Type = 1 to Types
   DMATIN(MAT, J) = DMATIN(MAT, J) + TMATIN1(Type, MAT; I)∗CAPF(Type, J) +
   + TMATIN2(Type, MAT; I)
  end Type
 end MAT
 if J = 1 then do
  DUNATT(1) = DUNATT(1) + DMATIN(1, 1)
  DTHT(1) = DTHT(1) + DMATIN(2, 1)
 end if
```

```
 else do
  DUNATT(J) = DUNATT(J-1) + DMATIN(1, J)
  DTHT(J) = DTHT(J-1) + DMATIN(2, J)
 end else
```

REFABR(J) = DMATIN(9, J)/BT	Annual re-fabrication amount (t HM/a) in period J
DUNAT(J) = DMATIN(1, J)	Consumption of natural U (U_{nat}) in period J
DUNATT(J)	Cumulated consumption of U_{nat} up to period J (tonnes)
DTHT(J)	Cumulated consumption of Thorium up to period J (tonnes)

```
 Do MAT = 10 to 15
  FACTOR = 0
  Do Type = 1 to Types
        If (I-ICT(Type)) > 0 then FACTOR = FACTOR +
        + TMATOUT1(Type, MAT, I-ICT(Type)) ∗CAPF(Type, J) + TMATOUT2(Type,
          MAT; I-ICT(Type))
  end Type
  if J = 1 then DSTORAGE(MAT, 1) = DSTORAGE(MAT, 1) + FACTOR
  else DSTORAGE(MAT, J) = DSTORAGE(MAT, J-1) + FACTOR
  If MAT > 10 then DSTORAGE(MAT, J) = DSTORAGE(MAT, J) – DMATIN(MAT-8, J)
 end MAT
```

MAT = 16:

```
 if J = 1 then DSTORAGE(16, 1) = DSTORAGE(16, 1) + FACTOR – DMATIN(7, 1)
 else DSTORAGE(16, J) = DSTORAGE(16, J-1) + FACTOR – DMATIN(7, J)
 FACTOR = 0
  Do Type = 1 to Types
        If (I-ICT(Type)) > 0 then FACTOR = FACTOR +
        + TMATOUT1(16, I-ICT(Type)) ∗CAPF(Type, J) + TMATOUT2(16, I-ICT(Type))
  end Type
 DSTORAGE(16, J) = DSTORAGE(16, J) + FACTOR – DMATIN(8, J)
end J
```

4.3 Computation of the waste amounts

The waste comprises (as above for the coarse time step) the cumulated discharged (spent) amounts of

- Th, FPs, non-re-fabricated amounts of U, MA, and PU, and
- re-fabrication losses of losses of U, Pu, and MA

```
DUIRRwaste(1) = RESID-value
DPUwaste(1) = RESID-value
DMAwaste(1) = RESID-value
```

VPU, VUIRR, VMA [1]: Loss factors at reprocessing of U-irr, Pu and MA (see first page)

```
Do J = 1 to N
 If J = 1 then do
   DUIRRwaste(1) = DUIRRwaste(1) + VUIRR∗DMATIN(5, 1)
   DPUwaste(1) = DPUwaste(1) + VPU∗(DMATIN(7, 1) + DMATIN(8, 1))
   DMAwaste(1) = DMAwaste(1) + VMA∗DMATIN(6, 1)
 end if
 else do
   DUIRRwaste(J) = DUIRRwaste(J-1) + VUIRR∗DMATIN(5, J)
   DPUwaste(J) = DPUwaste(J-1) + VPU∗(DMATIN(7, J) + DMATIN(8, J))
   DMAwaste(J) = DMAwaste(J-1) + VMA∗DMATIN(6, J)
 end else
 DWASTE(J) = DSTORAGE(10, J) + DSTORAGE(11, J) + DUIRRwaste(J) + DPUwaste(J)
                + DMAwaste(J)
end J
```

List of the required RESID-values (initial values at the beginning of the first period):

DSTORAGE(MAT, 1) = STORAGE(MAT, 1) = RESID-values for MAT = 10 to 16
DUIRRwaste(1) = UIRRwaste(1) = RESID-value
DPUwaste(1) = PUwaste(1) = RESID-value
DMAwaste(1) = MAwaste(1) = RESID-value

References:

/NEA 2002/: NEA-OECD: Accelerator-driven Systems (ADS) and Fast Reactors (FR) in Advanced Nuclear Fuel Cycles, A Comparative Study, OECD 2002, http://www.nea.fr/html/pub/webpubs/welcome.html#ndd

/NERAC 2002/: Nuclear Energy Research Advisory Committee: A Technology Roadmap for Generation IV Nuclear Energy Systems, Technical Roadmap Report, December 2002, GIF-002-00, http://www.nuclear.gov/