SPRINT-S

S_trictly P_arallelized R_egional I_ntegrated N_umeric T_ool for S_imulation

Primer for Version 2.11 (May 1999)

Michael Flechsig
Potsdam Institute for Climate Impact Research
Telegrafenberg, D-14473 Potsdam, Germany
Phone: ++49 - 331 - 288 2604
Fax: ++49 - 331 - 288 2600
e-mail: flechsig@pik-potsdam.de

Abstract

SPRINT-S (S_trictly P_arallelized R_egional I_ntegrated N_umeric T_ool for S_imulation) is a parallelization tool at the experiment level for arbitrary simulation models. Validation, evaluation, and application of simulation models normally result in running the model independently, but in a coordinated way, several times. The same problem arises in relation to regional integrated modeling for laterally decoupled model designs. SPRINT-S supports these tasks by pre-defined simulation experiments like behavioural, sensitivity, perturbation and stochastic analysis. The experiments as well as spatial applications of site-related models are performed in parallel without a parallelization of the model itself. Result processing allows to navigate within result spaces.

SPRINT-S was developed for IBM’s scalable parallel machine SP2 using IBM’s proprietary Message Passing Library MPL. It can be adapted to any other parallel machine that supports the generic Message Passing Interface MPI or a cluster of networking machines by migrating from MPL to MPI.

Contents

Appendices

Figures and Tables

Document Conventions

• *italic*	marks a SPRINT-S subprogram, function or script
• < ... >	is a placeholder for a string
• { ... }	braces enclose an optional element
• [... | ... | ...]	brackets enclose a list of choices, separated by a vertical bar
• nil	stands for the empty string (nothing)
• `monospace`	indicates a programming example

1 Introduction

Computer based simulation involves experimenting with a model that is constructed to map features of a real world system under investigation to a computer. The process of model validation (adjusting the model behaviour to the real system) as well as many simulation studies (applying a validated model) often result in the demand for running a model independently but in a coordinated way several times. Such experiments are known as multi-run experiments. Examples for applying this technique are

- scanning the model behaviour dependent on model parameter and / or initial value settings
- determining model sensitivity from model parameters and / or initial values
- performing a state-deterministic model in a Monte Carlo experiment or a stochastic analysis

Earth system analysis is forced to experiment with models. Since many models of this scientific discipline are designed in a geographically explicit manner by neglecting lateral fluxes, an overall model of a geographic region can be considered for that case as a sequence of independent submodel runs. Each submodel run is responsible for determining the states of a certain geographic subspace under its appropriate site conditions. Agricultural patch models and forest gap models are typical representatives of this class of models. Even hydrological runoff models can be performed in this way, if the geographic subspaces under consideration are defined as river catchments.

Design and performance of multi-run experiments and spatially explicit simulation can be a very time consuming process: simulation experiment preparation and processing results by structuring and aggregating simulation results in space and in time are expensive tasks. The simulation time of the experiment itself can be a limiting factor with respect to the feasibility of the experiment.

The software tool described in this paper is devoted to support solving the methodological problems addressed above. First of all, SPRINT-S is a parallelization tool for any model at the experiment level. A sequence of independent model runs can be parallelized by using the parallel structure of SPRINT-S in a flexible manner without parallelizing the model itself. Generally speaking, the term model here means any algorithm that transforms an input vector to an output vector. It covers temporal and spatial dynamic models in the usual sense as well as static models and any algorithm with the transformation described above. Additionally, SPRINT-S supplies a set of pre-defined multi-run experiment types, ranging from experiments for model validation and usage to spatially distributed model applications and combinations of both techniques. So the tool also supports experimenting with models and algorithms by supplying experiment techniques to the user. SPRINT-S result processing makes it possible to navigate within result spaces and offers software interfaces for result transformation.

SPRINT-S was developed for IBM's scalable parallel machine SP2 using IBM's proprietary Message Passing Library MPL (IBM, 1994). It can be adapted to any other massively parallel machine or a cluster of networking machines by migrating from MPL to the generic Message Passing Interface MPI.

2 Overview on Experiment Types

SPRINT-S supplies a set of pre-defined multi-run experiment types, where each type addresses a special experiment class for performing a simulation model or any algorithm with an input - output transition behaviour. In the following, experiment types will be described for time dynamic simulation models, because this class forms the majority of SPRINT-S applications. All information can be transformed easily to any other algorithm.

Based on systems' theory, each time dynamic model M can be formulated - without limitation of generality for the time discrete and state deterministic case - as

$$Z(t) = ST\ (\ Z(t-\Delta t)\ ,\ \ldots\ ,\ Z(t-n*\Delta t)\ ,\ X(t)\ ,\ Z(t_0)\ ,\ P\ )$$

with	ST	state transition description
	Z	state vector
	X	input vector
	P	parameter vector
	t	time
	Δt	time increment
	t_0	initial time step
	n	time delay

The output vector Y is a function of the state vector Z:

$$Y(t) = OU\ (\ Z(t)\ ).$$

In the following, z, x, p, and y are components of the vectors Z, X, P, and Y, respectively.

So model behaviour Z is determined for fixed t_0, n and Δt by state transition description ST, driving forces X, initial values $Z(t_0)$, and parameters P. Manipulating and exploring model behaviour in any sense means changing these four model components. While state transition description ST reflects mainly model structure and is quite complex to change, each component of the driving forces vector X normally is a time-dependent vector. Initial value and parameter vector components are determined by single values.

Introduction of additional technical parameters P_t can reduce the complexity of handling a model with respect to the four model components, described above. Changes in state transition description ST can be pre-determined in the model by assigning values of a technical parameter to alternative submodel versions, which are switched on or off by the technical parameters. Each component of the driving forces vector X can be combined with technical parameters in different ways:

- by selecting special driving forces dependent on the technical parameter value
- by manipulating the driving forces with the parameter value (e.g., as an additive or multiplicative increment)
- by parametrizing the shape of a driving force

When this is done, the model behaviour finally depends only on the parameters P and the initial values $Z(t_0)$. From the methodical point of view there is no difference between parameters and initial values, because both they are constant during one model run. That is why in the following discussion of experiment types a parameter stands as a placeholder for all the four model components.

2.1 Behavioural Analysis

Behavioural analysis is the inspection of the model's behaviour in a space that is spanned up by selected model parameters. Pre-defined values P_i are assigned to selected components of the vector P. { Z(t) } as the dynamics of Z(t) over all model runs of the formed run ensemble then depend on the constellation of P.

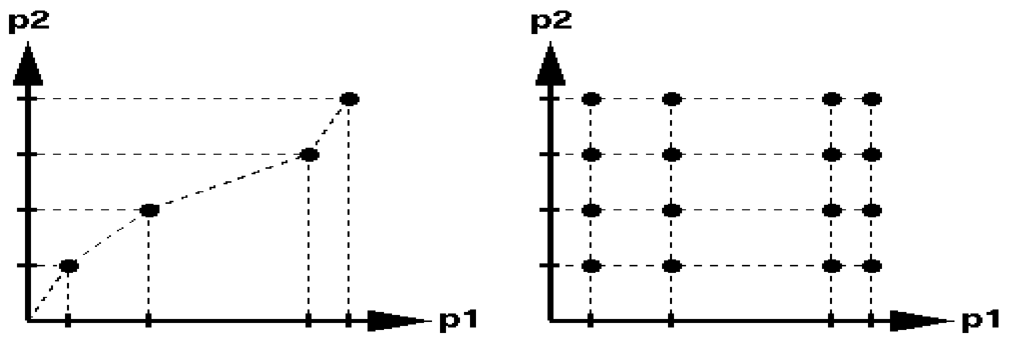
$$\{ Z(t) \} \quad = \quad \text{run\_ensemble} (ST , (P_1, P_2, ..., P_k))$$

This model inspection can be interpreted and used in different ways:

- for scenario analysis:
  to show how model behaviour changes with changes of parameter values
- for numerical validation purposes:
  to determine parameter values in such a way that the output vector matches with measurement results of the real system
- for deterministic error analysis:
  to analyse how the model error is dependent on parameter errors
- for a simulation-based control design:
  to determine parameter values in such a way that a goal function becomes an extremum

SPRINT-S behavioural analysis is a generalization of the one-dimensional case, where the model behaviour is scanned in dependence on the deterministic change of a parameter value. The n-dimensional case with n parameters demands a strategy for scanning n-dimensional spaces in a flexible manner. On the basis of the SONCHES simulation environment (Wenzel et al., 1990) subspaces of the n-dimensional investigation space can be scanned on the subspace diagonal (parallely in a one-dimensional hyperspace) or completely for all dimensions (combinatorially in a lattice-like approach) and both techniques can be combined. Besides this regular scanning method an irregular, file-based technique is possible.

The resulting number of single simulation runs for the experiment depends on the number of parameter adjustments per dimension together with the selected scanning method. An experiment is described by the names of the involved parameters, their increments and their combination (scanning method). Result processing resolves the scanning method again and outputs results as projections on one- or two-dimensional parameter subspaces.



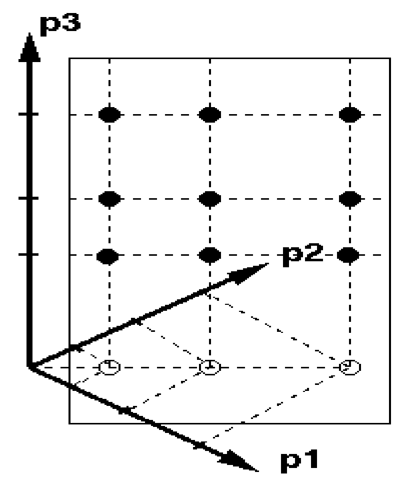


Fig.1 describes the regular scanning technique by an example. In the upper left scheme the two-dimensional parameter space for the parameters p_1 and p_2 is scanned in parallel (in SPRINT-S syntax as p_1,p_2) by 1+1+1+1 = 4 model runs, while the upper right scheme represents a combinatorial scanning of (p_1*p_2) with 4*4 = 16 model runs. For the lower scheme the complex scanning strategy of the 3-dimensional parameter space is (p_1,p_2)*p_3 with (1+1+1)*3 = 9 model runs. Each filled dot represents a single model run.

Fig. 1 Scanning of high-dimensional parameter spaces

2.2 Sensitivity Analysis

Sensitivity analysis serves to determine sensitivity functions. In classical systems' theory, model sensitivity of z with respect to p is the partial derivative of z after p. In the numerical simulation of complex systems, finite sensitivity functions are preferred, because they can be obtained without model enlargements or re-formulations. They are linear approximations of the classical model sensitivity.

$$\{ Z(t) \} \quad = \quad \text{run\_ensemble} \; (\; ST \; , \; P \pm \Delta P \;)$$

Sensitivity functions can be used for localizing modification-relevant model parts as well as control-sensitive initial values in control problems. On the other hand, identification of robust parts of a model or even complete robust models makes it possible to run a model under internal or external disturbances.

Sensitivity analysis in SPRINT-S is based on finite sensitivity functions, which are defined as follows:

linear	$\text{LIN} = \dfrac{z(p \pm \Delta p) \; - \; z(p)}{\Delta p}$
squared	$\text{SQR} = \dfrac{(\; z(p \pm \Delta p) \; - \; z(p) \;)^2}{\Delta p}$
absolute	$\text{ABS} = \dfrac{\mid z(p \pm \Delta p) \; - \; z(p) \mid}{\Delta p}$
relative	$\text{REL1} = \dfrac{z(p \pm \Delta p) \; - \; z(p)}{z(p) * \Delta p}$
relative	$\text{REL2} = \dfrac{\dfrac{z(p \pm \Delta p) - z(p)}{z(p) * \Delta p}}{\dfrac{\Delta p}{p}}$
symmetry test	$\text{SYM} = \dfrac{z(p + \Delta p) \; - \; z(p - \Delta p)}{\Delta p}$

So the sensitivity of the model to a parameter is always expressed as the sensitivity of a state variable z, usually at a selected time step within a surrounding Δp of a parameter value p. That is why the conclusions drawn from a sensitivity analysis are only valid locally with respect to the whole parameter space. Additionally, sensitivity functions only describe the influence of one parameter p of the whole vector P on the model's dynamics.

Linear, squared and absolute sensitivity functions allow comparison of the influence of various parameters on the same state variable. The relative sensitivity functions are suited to comparing the sensitivity of the same parameter on different state variables, because of the normalization effect of the nominal state variable z(p) and the nominal parameter value p for the two relative sensitivity functions. The symmetry test will return zero if the state variable z shows a symmetrical behaviour in the surrounding of the nominal value of p.

A sensitivity experiment is described by the names of the parameters p to be involved and the increments Δp. The number of runs for the experiment results from the number of parameters and increments: two runs per parameter for each increment plus the nominal run. Sensitivity functions are calculated during result processing.

2.3 Perturbation Analysis

A perturbation analysis in SPRINT-S is a Monte Carlo simulation with pre-run parameter perturbations.

Theoretically, with a Monte Carlo analysis moments of state variables can be computed as

$$M^{(k)}\{z\} = \int_{P_m} \dots \int z(p_1,...p_m)^k * pdf(p_1,...,p_m)\, dP_m$$

with

$P_m = (p_1,...,p_m)$	parameter subspace for which the Monte Carlo analysis will be performed
$M^{(k)}\{z\}$	k-th moment of the state variable z with respect to the probability density function f
$z(P_m)$	state variable z as function of parameters P_m
$pdf(P_m)$	probability density function of parameters P_m

By interpreting the probability density function $pdf(p_1,...,p_m)$ as the error distribution of the parameters P_m it is possible to study error propagation in the model. On the other hand perturbation analysis can be interpreted as a stochastic error analysis, if there are measurements of the real system for z.

For a numerical simulation it is assumed that the probability density function $pdf(P_m)$ can be decomposed into independent probability density functions pdf_i for all parameters p_i of P_m:

$$pdf(p_1,...p_m) = \prod_{i=1}^{m} pdf_i(p_i)$$

and the m-dimensional integral is approximated by a sequence of simulation runs where the parameter values of p_i are perturbed according to the probability density function f_i.

$$\{ Z(t) \} = \text{run\_ensemble} (ST , \text{pdf}(P_m) \text{ for disturbing } P_m)$$

On the basis of these assumptions, the following statistical measures are computed during performance of a perturbation analysis with n simulation runs and realizations $(z_1, ..., z_n)$ of the state variable z:

minimum	$z^{(min)}$	$= \min_{i=1,n} (z_i)$
maximum	$z^{(max)}$	$= \max_{i=1,n} (z_i)$
sum	$z^{(sum)}$	$= \sum_{i=1}^{n} z_i$
mean $(M^{(1)})$	$z^{(1)}$	$= \sum_{i=1}^{n} z_i / n$
variance $(M^{(2)})$	$z^{(2)}$	$= \sum_{i=1}^{n} (z_i - z^{(1)})^2 / (n - 1)$
skewness $(M^{(3)})$	$z^{(3)}$	$= \sum_{i=1}^{n} (z_i - z^{(1)})^3 / n$
confidence boundary	$co^{(\alpha)}$	$= t_{\alpha,n} \sqrt{z^{(2)} / n}$ $\quad t_{\alpha,n}$: significance boundaries of Student distribution
class frequency	$cl^{(class)}$	$=$ number of z_i with $class_{min} \leq z_i < class_{max}$

Confidence interval boundaries are $z^{(1)} \pm co^{(\alpha)}$ for a level of error α = 1% and 5%, respectively. The confidence interval is only valid if z is normally distributed. Class frequencies can be plotted as heuristic probability density function of z.

The following probability density functions for parameters to be perturbed are pre-defined in SPRINT-S:

distribution	probability density function	distribution parameters
uniform	$\text{pdf}(x) = \frac{1}{b-a}$ if $x \in [a,b]$ $\text{pdf}(x) = 0$ otherwise	a lower boundary b upper boundary a < b It is: mean = (a+b) / 2 variance = $(b-a)^2 / 12$
normal	$\text{pdf}(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$	μ mean σ standard deviation $\sigma > 0$
lognormal	$\text{pdf}(x) = \frac{1}{x\sigma\sqrt{2\pi}} \exp\left(-\frac{(\ln x-\mu)^2}{2\sigma^2}\right)$ if $x > 0$ $\text{pdf}(x) = 0$ otherwise	μ mean σ standard deviation $\sigma > 0$ It is: ln x is normally distributed with μ and σ
exponential	$\text{pdf}(x) = \frac{1}{\mu} \exp\left(-\frac{x}{\mu}\right)$ if $x > 0$ $\text{pdf}(x) = 0$ otherwise	μ mean $\mu > 0$ It is: variance = μ^2

Tab. 1 Probability density functions

The number of runs to be performed during an perturbation analysis has to be specified. An experiment is described by the parameters involved in the analysis, their distribution and the appropriate distribution parameters.

2.4 Stochastic analysis

Stochastic analysis is a Monte Carlo analysis with parameter perturbations per time step. Other than with the perturbation analysis, parameters can be adjusted each time step according to the selected distribution.

$$\{ Z(t) \} \quad = \quad \text{run\_ensemble} \ (\ ST \ , \ \text{pdf}(P_m) \text{ for disturbing } Z(t) \ )$$

This allows stochastic investigation of the model in two ways: If the perturbed parameter

- is a model parameter, related to a process or a submodel, then the stochastic analysis maps stochastic influences of the perturbed parameter onto the process or the submodel
- is used as an additive noise to a rate equation of a time-discrete model, then intrinsic fluctuations for the total model behaviour can be studied

Pre-defined distributions and statistical measures available during result processing are the same as for the experiment type perturbation analysis.

2.5 Spatial Analysis

Application of a site-related model within a spatial / regional experiment normally leads to the problem of how to assign site-specific parameters for each site to the model and how to perform the model with the assigned data in a sequential manner (King, 1990, Flechsig et al., 1994). This experiment type has a lot in common with a behavioural analysis. The only difference arises from the structure of the site-specific data. Normally, they do not follow a regular schema which allows the inspection of the site parameter space for all sites in a straigthforward manner. Nevertheless, each site can be addressed by some geographic information, e.g., longitude and latitude or an identification number. It is more convenient to control a spatial application by this identification than by parameter data set of the whole site.

Spatial analysis uses this approach and combines the advantages of behavioural analysis with the possibility to derive site-related data sets from any spatial identification. This is the only experiment type where the input parameter set for the model can be derived from other information (normally the identification) before running the model itself. Performing this transformation within the model is an alternative, but limits its application because of the parallel structure of SPRINT-S (see the chapter on SPRINT-S parallel architecture) and because of the special layout of complex experiments (see the chapters on complex experiments).

Pre-model run transformation has to be defined in a user subprogram, which is described in detail in the chapter on software interfaces to the user model. For experiment performance see experiment type behavioural analysis.

2.6 Complex Experiments

The complex experiment combines the method of applying a site-related model spatially (the spatial subexperiment) with the possibility to run a behavioural, sensitivity, perturbation, or stochastic analysis (the non-spatial subexperiment) for all sites.
In this way, the complex experiment enlarges SPRINT-S applicability tremendously. So it is possible, e.g.,

- to study complex regional model behaviour for site-related model parameter adjustments (spatial and behavioural experiment types)
- to estimate regional yields, which are expected under a perturbed regional climate (spatial and stochastic experiment types)

One important feature of complex experiments is the possibility to include into the parameter set of the non-spatial subexperiment parameters of the spatial subexperiment. So called associations between parameters of the spatial and non-spatial subexperiment define new default parameter values for the non-spatial subexperiment from parameter values of the current site. So it is possible to perform for all sites of a region a perturbation analysis, where a parameter for the assigned probability density function is a componente from the site-related data set.

Perturbation and stochastic analyses can be performed in a complex experiment with local seeds for the random number generator, which are site-related or a global seed, which is valid for all sites.

3 SPRINT-S Parallel Architecture

SPRINT-S defines for an experiment a communication node, where all input, output, and preparation of the whole experiment as well as preparation of each single run of the run ensemble is performed. Beside this node a number of simulation nodes, which are responsible for performing single simulation runs are available and are assigned to SPRINT-S before starting it. CPU consumption for running an experiment scales linearly with the number of assigned simulation nodes.

Coupling a model to SPRINT-S and running a multi-run experiment need some preparations, which are described in the following:

First of all, the user model should be analysed with respect to model parts, which are performed independently on the actual model constellation, defined by parameter values or initial values at model startup. Such model parts always lead to identical results, which normally do not change when running the model. A typical example for such a „static pre-model“ is input of information from an external file by reading the file. Performing this pre-model for each single simulation run of the run ensemble is redundant and leads to a loss of performance efficiency of the experiment as well as to multiple concurrent access to external files. This part of the model should be performed once during experiment preparation on the communication node. A subroutine *prepare* serves as an interface, where the pre-model can be implemented and the results of the pre-model can be broadcast to all simulation nodes, where they are at disposal for running the model itself.

As a next step the remaining user model has to be „wrapped“ in a software interface for coupling it to SPRINT-S. The user model itself transforms input data (in correspondence with the chapter on the overview of experiment types this includes the model parameters, technical parameters, and initial values) into output data (the simulation results, which can be organized in any way in which the modeller is interested). The shell in which the user model has to be implemented, is the subroutine *simulation*. Normally the following tasks are performed in this subroutine:

- map input data of *simulation* to user model parameters, technical model parameters or initial values
- compute the user model
- collect model output result data and assign it to the output data of *simulation.*

SPRINT-S was written in FORTRAN. Both the user model and the identified pre-models can be written in FORTRAN or C.

Experiments are defined by so-called experiment description files (EDF). EDFs are structured ASCII files with

- the name of the experiment type
- symbolic (external) names of model parameters or initial values with their default values and experiment-specific additional values
- general experiment-specific information

As a general rule, the symbolic names of parameters or initial values should be the same as the internal names in the user model. The corresponding parameter values and initial values form the input data to the subroutine *simulation* and are mapped to the user model parameters, technical parameters and initial values.

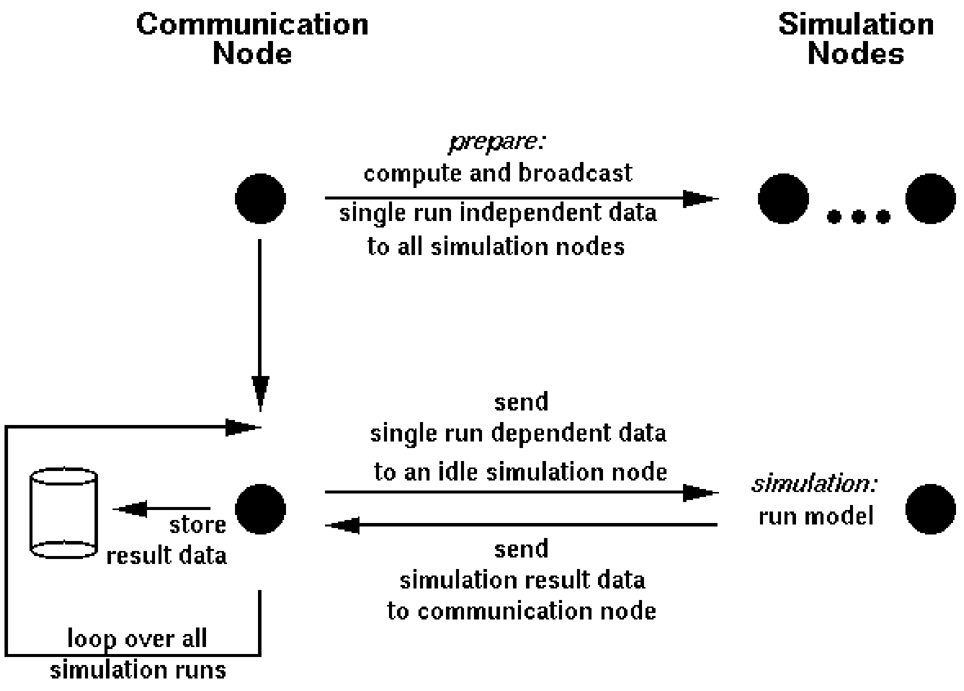


Fig. 2 Model performance in the parallel simulation environment

According to Fig. 2, performance of an experiment starts with *prepare* on all nodes and within *prepare* with the performance of the pre-model on the communication node: This is followed by a loop over all simulation runs on the communication node dependent on the EDF, where the communication node sends input data to an idle simulation node, which performs simulation and sends output data to the communication node. All the output data of *simulation* is gathered from SPRINT-S in a direct access output file for later result processing.

4 Software Interfaces to the User Model

4.1 Experiment Preparation Interface - the Subroutine *prepare*

Besides setting model-specific variables, the subroutine *prepare* has to fulfill one important purpose:
Often models need external data, independent of their special calibration (e.g., for sites), which normally do not change when running the model (pre-model). In *prepare* the user can supply the model with this information (normally by reading it from external files) once and forever for all model runs. So he / she can avoid the situation in which this step is performed indepenently for all model runs, with the risk of multiple concurrent access to external files and a connected loss of performance.

subroutine *prepare (at_comm_node)*

with

at_comm_node	logical*4, input to *prepare*	
	= .true.	if the current node is the communication node
	= .false.	else

The following rules apply to *prepare*

- The user always has to declare a subroutine *prepare*.
- *prepare* is performed at the beginning of a SPRINT-S experiment once at all nodes. The argument at_comm_node can be used to perform pre-models only at the communication node.
- At least the number of output data from *simulation* length_output_data has to be fixed within *prepare*. To do this, the user has to call at all nodes the SPRINT-S

  subroutine *put_length_output_data* (length_output_data)

  with

length_output_data	integer*4, input to *put_length_output_data* determines length of output vector output_data from subroutine *simulation*

  put_length_output_data can only be called within *prepare*.

- To address the problem discussed above of pre-models and redundant external data sharing among different model runs, SPRINT-S supplies the subprograms

  subroutine *broadcast* (field_to_broadcast , length_field_to_broadcast)
  subroutine *synchro*

  with

field_to_broadcast	integer*4 or real*4, input to *broadcast* field to broadcast from the subroutine *prepare* at the communication node to a field with the same name in the subroutine *simulation* or its subprograms on all simulation nodes
length_field_to_broadcast	integer*4, input to *broadcast* length of this field

  With *broadcast*, information read / computed in *prepare* at the communication node (at_comm_node = .false.) can be broadcast from the communication node to all simulation nodes to avoid multiple performance of information access. Note the following important restrictions:
  - The field_to_broadcast must be defined in *prepare* as well as in the simulation model *simulation* or its subprograms. One way to ensure this is to use common blocks for FORTRAN.
  - *broadcast* and *synchro* can be used only in *prepare*.
  - After broadcasting all information to the simulation nodes where the simulation model *simulation* resides, the user has to ensure that the received information will be unchanged. This is because the user will normally broadcast only such information as is unique to all simulation runs.
  - The broadcast information is received automatically at the simulation nodes / the simulation model *simulation*. Nothing has to be done by the user.
  - In *prepare* a block with broadcasts has to be bounded by *synchro* calls:

    ```
    call synchro
    call broadcast ( ... )
    ...
    call broadcast ( ... )
    call synchro
    ```

    broadcast and *synchro* have to be performed at all nodes.
- Within *prepare* the current experiment type can be made available by the SPRINT-S

  integer*4 function *iget_exp_type* ( )

  to distinguish between different experiment types and so to design only one preparation module *prepare* for different experiment types. For the resulting function value of *iget_exp_type* see Appendix A.

- For the spatial experiment type the length of the derived target vector length_derived_data has to be fixed by calling the SPRINT-S subroutine

  subroutine *put_length_derived_data* (length_derived_data)

  For more information see the spatial experiment type.

For examples of how to construct *prepare* see Appendix F.

4.2 Simulation Model Interface - the Subroutine *simulation*

To handle any simulation model within SPRINT-S the model has at least

- to be extracted from all the parts where - independent of the model run - information is read in / computed. These parts should be performed within the subroutine *prepare* and results should be broadcast to the simulation model. See also Appendix F.
- to be wrapped into a simulation subroutine with the name *simulation* for one model run, where
  - the current input data vector input_data which is derived from the EDF is mapped to the real model parameters / initial values
  - the model result data the user wants to postprocess during result processing are mapped from model variables to an output data vector output_data. For how to arrange model result data in the output data vector output_data, see Appendix F.

The appropriate subroutine has the following arguments:

**subroutine *simulation* (input_data , length_input_data ,
output_data , length_output_data)**

with

input_data	real*4, input to *simulation* target vector, derived from the EDF and transformed for the spatial experiment type Analysis by *spatial_transform*. To be mapped on real model parameters / initial values.
length_input_data	integer*4, input to *simulation* length of input_data Determined by the EDF or fixed for the experiment type spatial by calling *put_length_derived_data* in *prepare* do not change it

output_data	real*4, output from *simulation* output vector of simulation results to be gathered by SPRINT-S in a direct access file for later result processing
length_output_data	integer*4, input to *simulation* length of output_data Fixed in *prepare by* subroutine *put_length_output_data* do not change it

Within *simulation* the current experiment type can be made available by the SPRINT-S

integer*4 function *iget_exp_type* ( )

to distinguish between different experiment types and so to design only one simulation module *simulation* for different experiment types. For the return value of *iget_exp_type* see Appendix A.

For examples of how to build *simulation* see Appendix F.

5 Experiment Description File EDF

An EDF serves for experiment definition. Its general structure is

```
E: <exp_type_descr>
body_line 1
...
body_line n
E: <terminator1>
E: <terminator2>
```

5.1 EDF Rules

- EDFs are case-insensitive with the exception of any file names.
- Blank characters (spaces) as well as blank lines are ignored.
- EDF lines starting with a # as the first nonblank character are treated as comment lines.
- While for non-complex experiments at least one body line must be stated, for complex experiments EDFs without any body line are also possible.

- If <terminator2> is nil, the complete terminator2-line is obsolete.

5.2 EDF Body Lines for Non-complex Experiments

EDF body lines for non-complex experiments serve for definition and numerical realization of model targets, normally parameters and initial values of the model under investigation. The general structure of an EDF body line is

<adjustment_type> <target> : <def> { : <exp_spec_infos> }

with

<adjustment_type>	[A | M | S] with A for additive adjustment M for multiplicative adjustment S for value setting of <exp_spec_infos> to the <def> of <target>
<target>	symbolic parameter / variable name (max. 8 characters, otherwise truncated), normally corresponding to parameter / initial value names of the investigated model.
<def>	default value of <target> for the adjustment type M <def> must not be 0.
<exp_spec_infos>	information specific to the current experiment type

The sequence of EDF body lines corresponds

- for all but the spatial experiment type
  with the sequence of the field elements of input_data of the subroutine *simulation*
- for the spatial experiment type
  with the sequence of the field elements of origin_data of the subroutine *spatial_transform*.

5.3 EDF Body Lines for Complex Experiments

EDF body lines for complex experiments assign new default values to the non-spatial subexperiment. They are based on data derived from the spatial subexperiment. The general structure of a body line is

assoc <target_of_non_spatial_subexp> : <index_of_derived_data>

with

<target_of_non_ spatial_subexp>	symbolic parameter / variable name (max. 8 characters, otherwise truncated)

	from the non-spatial subexperiment
<index_of_derived_data>	index of the value in derived_data from the spatial subexperiment, which is to be used as the new default value of <target_of_non_spatial_ subexp>

For more information on complex experiments and EDF, see the chapters on the complex experiment type.

6 Experiment Type Descriptions

6.1 Behavioural Analysis

Goal	Inspection of model behaviour in parameter, initial value, driving forces subspaces
exp_type_descr	behaviour
exp_spec_infos	[<expl._increment_list> **|** <impl._increment_list> | nil] with <expl._increment_list> := <increment_1> , <increment_2> , ... , <increment_n> and <impl._increment_list> := <begin_value> (<increment_value>) <end_value>
terminator1	comb [default **|** <combination>] or file <file_name>
terminator2	nil

6.1.1 Adjustments

adjustment type	**S**	**A**	**M**
resulting value =	<incr>	<def> + <incr>	<def> * <incr>

6.1.2 EDF Examples

```
E: behaviour
A p1 : 1 : 1,2,3,4,5       resulting in values 2,3,4,5,6     for p1
M p2 : 2 : 1,2,3,4,5       resulting in values 2,4,6,8,10    for p2
S p3 : 3 : 1,2,3           resulting in values 1,2,3         for p3
S p4 : 4 : 1(2)6           resulting in values 1,3,5         for p4
E: comb default

E: behaviour               and the appropriate increment data file my.incr:    1  -4
A: p5 : 5                                                                     -1   4
M: p6 : 6                                                                      4  -1
E: file my.incr                                                               -4   1
```

6.1.3 The Combination

- The terminator1 **keyword comb** is allowed only for defined increment lists. Explicate increment lists can be continued on the next EDF line by using the tilde as the last character on the previous line.
- The **combination <combination>** defines the way in which the model parameter / initial value subspace spanned by the parameters / initial values in the EDF body lines, will be inspected by SPRINT-S: This is done by applying the operators „*****" and „**,**" to all targets of the EDF body lines.
- **The operator „,"** combines increments of different targets and so their resulting values parallely (on the diagonal).
  p1 , p2 of the above example results in combinations (2,2) , (3,4) , (4,6) , (5,8) , (6,10) for (p1,p2) and so in 5 runs.
  For the operator „," the targets must have the same number of increments.
- **The operator „*"** combines increments of different targets and so their resulting values combinatorially (for all mesh points).
  p3 * p4 of the above example results in combinations (1,1) , (1,3) , (1,5) , (2,1) , (2,3) , (2,5) , (3,1) , (3,3) , (3,5) for (p3*p4) and so in 9 runs.
- The operator „," has a higher priority than the operator „*".
  p1 , p2 * p3 , p4 of the above example results in combing combinatorially the 5 parallel combinations of (p1,p2) with the 3 parallel combinations of (p3,p4) and so in total in 15 runs.
- In the <combination> each target has to be used exactly once.
- By the default combination **comb default** all targets are combined combinatorially.
  comb default of the above example is equivalent to comb p1 * p2 * p3 * p4.
- The terminator1 **keyword file** is allowed only for undefined increment lists.
  The increments are then read from the ASCII **increment data file** <file_name>. All targets are assumed to be combined in parallel. Tabulator, comma and space are valid value separators in this external file, a sequence of separators is treated as a single separator. Each record of the data file represents one simulation run. The sequence of the increments in each record corresponds to the sequence of

the EDF body lines. Data file lines starting with a # as the first nonblank character as well as blank lines are treated as comment lines.
The above example results in combinations (6,-24) , (4,24) , (9,-6) , (1,6) for (p1,p2) and so in 4 runs. Combination is implicitly as comb: p1,p2.

6.1.4 Experiment Performance

- According to the terminator1 (keyword comb or keyword file) the appropriate runs are generated.
- The sequence of the runs, as they are performed and stored in the direct access file sprint.output, corresponds with the sequence of the increments in the ASCII file sprint.input. sprint.input is generated on the basis of terminator1.

6.2 Sensitivity Analysis

Goal	Finite model variable sensitivity analysis to deterministic parameter, initial value, driving forces changes with different sensitivity functions
exp_type_descr	sensitivity
exp_spec_infos	nil
terminator1	incr [<expl._increment_list> **|** <impl._increment_list>] with <expl._increment_list> := <increment_1> , <increment_2> , ... , <increment_n> and <impl._increment_list> := <begin_value> (<increment_value>) <end_value>
terminator2	nil

6.2.1 Adjustments

adjustment type	**S**	**A**	**M**
resulting value =	<def> ± <incr>	treated as M	<def> * (1 ± <incr>)

Note here that computation of resulting values differs from all other experiment types.

As an example, the linear sensitivity function (see the chapter on the introduction to the experiment types) is then as follows:

for adjustment S $\text{LIN} = \dfrac{z(\text{def} \pm \text{incr}) - z(\text{def})}{\text{incr}}$

for adjustment M $\text{LIN} = \dfrac{z(\text{def} * (1 \pm \text{incr})) - z(\text{def})}{\text{def} * \text{incr}}$

6.2.2 EDF Example

```
E: sensitivity
S p1 : 0.
M p2 : 2.
E: incr .01, .05
```

6.2.3 Experiment Performance

- Each target will be adjusted by the same increments as those stated in the EDF terminator1 line. For implicit increment definition see examples of behavioural analysis.
- The adjustment A is transformed automatically to M, indicated by a warning message to the file sprint.protocol.
- For finite sensitivity functions several runs have to be performed:
  - A run with the default values of the targets
  - Per target and per increment two runs with the default values of all targets except that one under consideration, where the adjustment is applied according to the above adjustment rules
  - So the number of resulting runs is
    2 * number_of_body_lines * number_of_increments + 1
- Results of each model run are stored and sensitivity functions are applied during result processing.
  The following sensitivity functions can be performed during result processing: Linear, squared, absolute, relative as well as a symmetry test.
- The sequence of the simulation runs and the sequence of the simulation results in sprint.output are determined in the following manner:

```
nominal run
loop   over increment sequence
       loop   over EDF body line sequence
              adjustment +
              adjustment -
       end loop
end loop
```

6.3 Perturbation Analysis

Goal	Pre-model run parameter, initial value, driving forces perturbation analysis (Monte Carlo analysis)
exp_type_descr	perturbation
exp_spec_infos	<distribution> { (<distr_param_1> , <distr_param_2>) }
terminator1	runs <number_of runs>
terminator2	nil

6.3.1 Adjustments

adjustment type	**S**	**A**	**M**
resulting distribution parameter =	<distr_param>	<def> + <distr_param>	<def> * <distr_param>

6.3.2 Distribution Functions and their Parameters

Distribution function	**<distri-bution>**	**resulting <distr_param_1>**	**resulting <distr_param_2>**	**restriction**
uniform	U	lower boundary	upper boundary	lower boundary < upper boundary
normal	N	mean value	standard deviation	standard deviation > 0
lognormal	L	mean value	standard deviation	standard deviation > 0
exponential	E	mean value	---	mean value > 0

Tab. 2 Distribution functions and their parameters

For more information on the distribution functions see the chapter on the overview on experiment types.

6.3.3 EDF Example

E: perturbation

S p1 : 1. : N(1., 0.4)	so, p1 is a realization of N(1. , 0.4)
M p2 : 2. : U(0.5 , 1.5)	so, p2 is a realization of U(1. , 3.)
E: runs 250	250 model runs will be performed

6.3.4 Experiment Performance

- The number of runs must be greater than 10.
- Firstly, a model run with the default values of the targets will be performed which represents the deterministic case.
- All other runs will be performed with target values which result from a realization of the selected distribution functions for all targets. They are random numbers with respect to the selected distribution function and its distribution parameters. Each random number is fixed for a single model run (= pre-run perturbations).
- During result processing extrema and moments (up to skewness) and confidence intervals as well as the deterministic case of the simulation output data vector can be output.
- The sequence of the simulation runs to be performed starts with the nominal run, followed successively by the perturbation runs. The sequence of the result records in the direct access result file sprint.output is as follows:
  nominal run
  minimum over all perturbation runs
  maximum over all perturbation runs
  mean value over all perturbation runs
  variance over all perturbation runs
  skewness over all perturbation runs
  distance of confidence boundaries from mean value for the 99% significance level
  distance of confidence boundaries from mean value for the 95% significance level
  sequence of perturbation runs

6.4 Stochastic Analysis

Goal	Analysis of model parameter, initial value, driving forces perturbations per time step
exp_type_descr	stochastic
exp_spec_infos	<distribution> { (<distr_param_1> , <distr_param_2>) }
terminator1	runs <number_of runs>
terminator2	calls <number_of_stochastic_calls_per_model_run>

6.4.1 Adjustments

See experiment type Perturbation Analysis

6.4.2 Distribution Functions and their Parameters

See experiment Perturbation Analysis

6.4.3 EDF Example

E: stochastic	
S p1 : 1. : N(1., 0.4)	so, p1 is a realization of N(1. , 0.4)
M p2 : 2. : U(0.5 , 1.5)	so, p2 is a realization of U(1. , 3.)
E: runs 250	250 model runs will be performed
E: calls 23	there are 23 calls of *stochastic* per model run

6.4.4 Experiment Performance

- The number of runs must be greater than 10.
- Stochastic analysis can be applied to the model for model parameters where the parameter value is a random number that is changed per time step as well as for state variables where an additive additional noise term can be applied during each time step.
  To apply these techniques to the model for this task, the model has to be changed before running SPRINT-S. SPRINT-S supplies the user with a

  real*4 function *stochastic* (EDF_body_line)

  with

EDF_body_line	integer*4, input to stochastic number of the body line, the target is to be perturbed.

  which generates a random number with respect to the distribution function of the appropriate body line target and its distribution parameters.
- <number_of_stochastic_calls_per_model_run> is the total number of calls of the function *stochastic* per model run which has to be determined before running the

model to ensure the independence of the random numbers of the distribution functions for the single model runs.

- Firstly, a model run will be performed representing the deterministic case, where the function *stochastic* returns the default value of the target.
  If *stochastic* is used as an additional noise term, it should return for this case with 0., which corresponds with a default value of 0. in the appropriate EDF body line.
- All other runs will be performed with target values, which result in a realization of the selected distribution functions for all targets: They are random numbers with respect to the selected distribution function and its distribution parameters. Each target realization is computed anew by calling the function *stochastic* (perturbations per time step).
- Note that this is the only experiment where input_data of subroutine *simulation* cannot be exploited. Information transfer is based only on usage of the *stochastic* subroutine.
- During result processing, extrema and moments (up to skewness) and confidence intervals as well as the deterministic case of the simulation output data vector can be output.
- For the sequence of the simulation runs to be performed and for the sequence of the results in the direct access result file sprint.output, see experiment type Perturbation Analysis.

6.4.5 Example for Using *stochastic*

A model without stochastic perturbations could appear as follows

```
...
do itime = 1 , 100
      ...
      state1 = state1 + p1 * rate1
      state2 = state2 + p2 * rate2
      ...
enddo
...
```

With the EDF

E: stochastic	
S p1 : 1. : N(1., 0.4)	p1 is a realization of N(1. , 0.4)
A state2 : 0 : U(0.5 , 1.5)	state2 is a realization of U(0.5 , 1.5)
E: runs 250	perform 250 model runs
E: calls 200	= 100 time steps * 2 calls of *stochastic*

the model could be adopted to a stochastic experiment as follows

```
...
do itime = 1, 100
      ...
      state1 = state1 + stochastic(1) * rate1
      state2 = state2 + p2 * rate2 + stochastic(2)
```

```
            . . .
    enddo
    . . .
```

This results in a stochastic realization of p1 and an additive noise term to the variable state2.

6.5 Spatial Analysis

Goal	Spatial application of site-related models
exp_type_descr	spatial
exp_spec_infos	[<expl._increment_list> **|** <impl._increment_list> **|** nil] with <expl._increment_list> := <increment_1> , <increment_2> , ... , <increment_n> and <impl._increment_list> := <begin_value> (<increment_value>) <end_value>
terminator1	comb [default **|** <combination>] or file <file_name>
terminator2	nil

6.5.1 Adjustments

adjustment type	**S**	**A**	**M**
resulting value =	<incr>	<def> + <incr>	<def> * <incr>

6.5.2 EDF Examples

```
E: spatial
S lon : 0 : 1 (1) 360
S lat  : 0 : 1 (1) 180
E: comb default

E: spatial
S lon : 0
S lat  : 0
E: file land_mask.dat
```

6.5.3 The Combination

See experiment type Behavioural Analysis

6.5.4 Transformation of Target Values

When applying a site-related point model to a spatial / regional application without considering lateral fluxes, the problem of assigning site-specific parameters, initial values and driving forces for each site and performing the model with these data sets in a sequential manner normally occurs. From the methodological point of view this problem has a lot in common with a behavioural analysis. Nevertheless the situation remains data sets that are normally site-related cannot be formulated in such a compact manner as data supplied by the syntax calculus of the EDF for behavioural analysis. On the other hand the modeler is often able to assign to each site a unique identification, specified e.g. by a (longitude , latitude) pair or by an identification number. So it is more convenient to control the spatial application of the site-related point model by this identification rather than by the whole site-related data set.

When using this approach for spatial applications it remains the necessary to derive the appropriate site-related data set from the unique identification, or in terms of the EDF language calculus, to transform the original target values to derived model-related target values.

For the spatial analysis and for this experiment type in SPRINT-S only, the problem is solved by enabling the user to define a deviation / transformation subroutine

**subroutine *spatial_transform* (origin_data , length_origin_data ,
derived_data , length_derived_data)**

with

origin_data	real*4, input to *spatial_transform* original targets as defined in the EDF. Corresponds with the sequence on EDF body lines
length_origin_data	integer*4 , input to *spatial_transform* length of origin_data It is determined by the number of EDF body lines do not update it within *spatial_transform*
derived_data	real*4, output from *spatial_transform* This is the result of the transformation in *spatial_transform* It will be used as argument input_data of subroutine *simulation*

length_derived_data	integer*4, input to *spatial_transform* length of derived_data It is determined within *prepare* by the subroutine *put_length_derived_data* do not update it

The following figure sketches the information flow from the EDF to the vector output_data for all non-complex experiments.

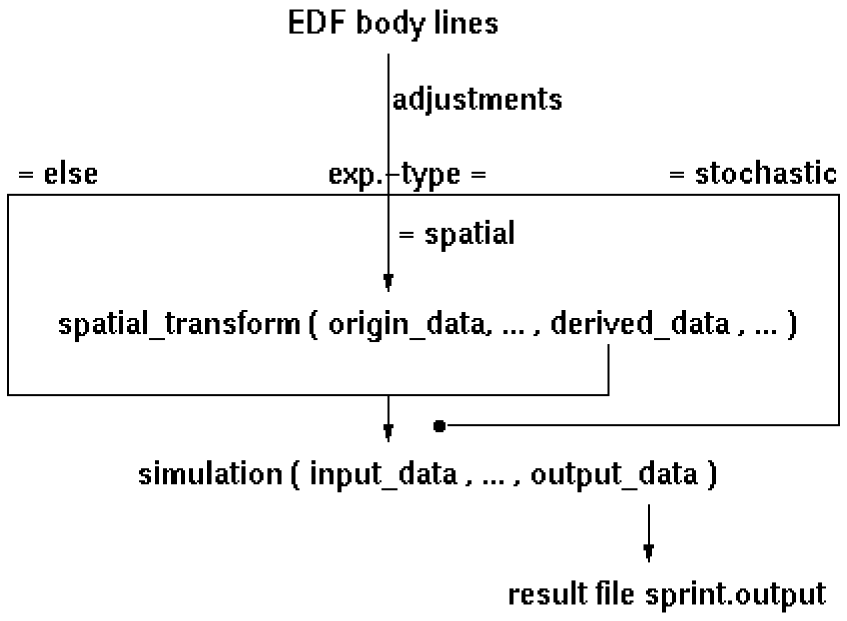

EDF body lines
adjustments
= else
exp.-type =
= stochastic
= spatial
spatial_transform (origin_data, ... , derived_data , ...)
simulation (input_data , ... , output_data)
result file sprint.output

Fig. 3 From EDF to output_data for non-complex experiments

The following rules apply in connection with spatial_transform

- Adjustments of the increments to the defaults are performed before running the subroutine ***spatial_transform*** , which also is indicated by the *spatial_transform* input argument origin_data.
- To determine length_derived_data the user has to perform within subroutine *prepare* at all nodes the SPRINT-S

subroutine *put_length_derived_data* (length_derived_data)

with

length_derived_data	integer*4, input to *put_length_derived_data* determines length of vector derived_data from subroutine *spatial_transform*

put_length_derived_data only can be performed within *prepare*. If *put_length_derived_data* is missing within *prepare*, the simulation experiment will be aborted.

- *spatial_transform* is performed on the communication node before sending run-specific information to a simulation node. So within *spatial_transform* the user can access external files without the risk of multiple concurrent access. These external files should be opened in the subroutine *prepare*.
- *spatial_transform* only will be used for the spatial experiment, also in the case that a user-defined subroutine *spatial_transform* is available for other experiment types.
- With an undefined *spatial_transform* information the experiment will be aborted (missing subroutine *spatial_transform*).
- The field derived_data of the subroutine *spatial_transform* is copied to the field input_data of the subroutine *simulation*.

6.5.5 Example for Using *spatial_transform*

A laterally uncoupled dynamic vegetation model should be performed globally at a 1° x 1° geometry as indicated above by the two EDFs.
Dependent on lon and lat for each land grid cell a parameter and initial value data set of 20 values has to be read from an external file. A land mask indicator, lon, lat and these 20 values are then used by the simulation model for one model run at (lon,lat). isite_data is assumed to be a user defined function.

```
      subroutine spatial_transform ( origin_data  , length_origin_data ,
     #                               derived_data , length_derived_data )
      integer*4 length_origin_data , length_derived_data
      real*4 origin_data (length_origin_data),
      real*4 derived_data (length_derived_data)

c     as indicated by the two EDFs  origin_data (1) = lon
c                                   origin_data (2) = lat
c     get now from the external file dependent on lon and lat
c     the site-specific data by a user defined function isite_data
c     and copy it directly to derived_data ( 4... ).
c     Open the external file in prepare.
c     ireturn = 0 :     record found, all o.k.
c     ireturn = 1 :     record not found, what means, that (lon,lat)
c                       corresponds with a grid cell outside the land mask
      ireturn = isite_data ( ‘read’ , origin_data (1) , origin_data (2) ,
     #                       derived_data (4) )
c     copy ireturn, lon, lat to the head of derived_data,
c     so length_derived_data = 23
      derived_data (1) = float ( ireturn )
      derived_data (2) = origin_data (1)
      derived_data (3) = origin_data (2)
      return
      end
```

6.5.6 Unfeasible Derived Data

If derived_data (1) = 1 the dynamic vegetation model in *simulation* must not be performed. While this will occur for the first EDF often because of the 360 * 180 runs for the total globe, this situation is avoided by the second EDF when the file land_mask.dat describes only those (lon,lat) constellations, the user is interested in. Nevertheless also for the case of derived_data = 1 *simulation* will be performed. The user has to ensure within *simulation* that the dynamic vegetation model will not be performed.

For an additional example of how to use spatial_transform see Appendix F.

6.5.7 Experiment Performance

See experiment type Behavioural Analysis

6.6 Complex Experiments

Goal	Perform per patch / site / grid cell of the spatial experiment type a behavioural, sensitivity, perturbation or stochastic analysis
exp_type_descr	spatial + [behaviour | sensitivity | perturbation | stochastic]
body_line	assoc <target_of_non-spatial_subexp> : <index_of_derived_data> (body lines are optional)
terminator1	files <EDF_spatial_subexp> + <EDF_non-spatial_subexp>
terminator2	seed [global | local] (only for non-spatial subexperiments perturbation and stochastic analysis)

Complex experiments combine the method of application of a patch- / site- related model spatially (the spatial subexperiment **subexp**) with the possibility of running a behavioural, sensitivity, perturbation, or stochastic analysis (the non-spatial subexperiment) in such a way that this analysis is performed for all patches / sites. Complex experiments enlarge SPRINT-S applicability tremendously by enabling scenario studies of spatially explicit models and supplying techniques in SPRINT-S result processing for navigating, handling and aggregating complex experiment output data.

For example, it is possible to use a crop model which depends on meteorology, soil, and management practices within a particular region to study

- the complex regional model behaviour for site-related model parameter adjustments or
- the expected regional yield (confidence intervals!) by perturbing meteorological input data and thus running a stochastic analysis for each crop production site.

6.6.1 EDF Syntax and Examples

The approach of combining a spatial analysis with another, non-spatial experiment is reflected in the EDF syntax: Each complex EDF combines a spatial EDF with a non-spatial EDF.

With the EDF file **spatial.edf** for 21 * 21 = 441 runs

```
E: spatial
S lon : 0 : 13 (0.1) 15
S lat  : 0 : 51 (0.1) 53
E: comb default
```

and the EDF file **behav.edf** for 5 runs

```
E: behaviour
M par1 : 1. : 0.9 (0.05) 1.1
A par2 : 2. : 0.1 , 0.25 , 0.2 , 0.05 , 0.15
E: comb par1 , par2
```

consider the EDF **complex1**

```
E: spatial + behaviour
E: files spatial.edf + behav.edf
```

which defines a complex experiment, where for all 0.1° x 0.1° grid elements a behavioural analysis will be performed. In total, the number runs of this experiment is 441 * 5 = 2205. A so-called **super-run** is related to the performance of a non-spatial subexperiment for one patch / site / grid cell of the spatial subexperiment. The experiment has 441 super-runs.

For the spatial subexperiment with the EDF spatial.edf a transformation / deviation of EDF targets lon and lat to site-specific parameters / initial values is performed by applying the user-defined subroutine *spatial_transform*. These site-specific values can be associated with targets of the non-spatial EDF to define new target default values, which are then also site-related. The optional body lines in the complex EDF serve this approach:

Consider the EDF **complex2**

```
E: spatial + behaviour
```

assoc par1 : 2
E: files spatial.edf + behav.edf

where the resulting value of index 2 (position 2) of the derived_data vector from *spatial_tranform* is used to replace the original default value 1. of par1. Normally the new default value of par1 will differ between the single grid elements. So it is possible to consider data which are derived by *spatial_transform* in the non-spatial experiment of complex experiments. The default value of par2 will not change in space.
For more information about data vector structures and about associated targets see below.

The **terminator2** line is only defined for the non-spatial subexperiments Perturbation and Stochastic Analysis. It addresses different possibilities in handling random numbers:

- While seed local ensures that for each perturbation / stochastic analysis the random number seed differs between all patches / sites / grid cells,
- seed global ensures usage of a unique seed for all perturbation / stochastic subexperiments.

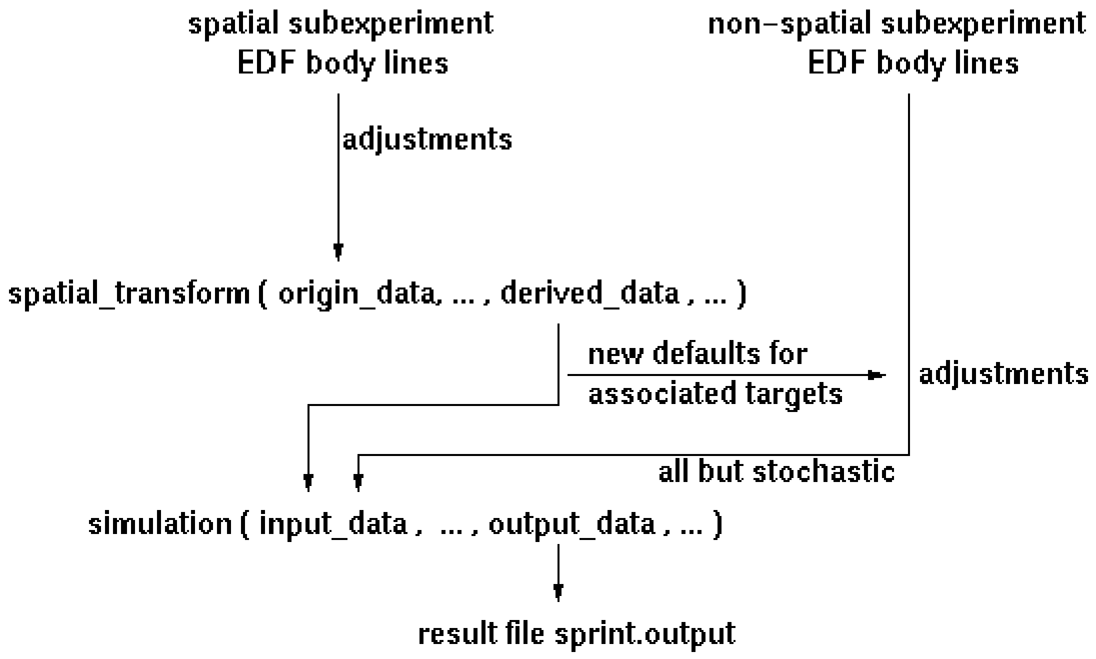

spatial subexperiment
EDF body lines
non-spatial subexperiment
EDF body lines
adjustments
spatial_transform (origin_data, ... , derived_data , ...)
new defaults for
associated targets
adjustments
all but stochastic
simulation (input_data , ... , output_data , ...)
result file sprint.output

Fig. 4 From EDFs to output_data for complex experiments

6.6.2 Complex Data Vector Structures and Associations

The following rules are valid in connection with complex experiments for the subroutines the user has to define, and for the application of SPRINT-S subroutines and functions:

- As one subexperiment is from type spatial analysis, subroutine *put_length_derived_data* has to be performed in subroutine *prepare*. The argument length_derived_data of subroutine *put_length_derived_data* is related only to the non-spatial subexperiment. This is consistent with the general purpose of the subroutine *spatial_transform*.
- The input data vector of the subroutine *simulation* is composed from the output vector derived_data from the subroutine *spatial_transform*, followed by the adjusted target values of the non-spatial EDF.

As mentioned above, associations define new site / patch- / grid cell-related default values for targets from the non-spatial subexperiment. In the body line the <index_of_derived_data> is the position in the vector derived_data of the subroutine *spatial_transform* where the new default value of the associated target of the non-spatial experiment is located.

6.6.3 Association Example

To illustrate associations, consider complex EDF complex2 described above in connection with a rather synthetic subroutine *spatial_transform* as

```
...
derived_data (1) = origin_data (2) + origin_data (1)       = lat + lon
derived_data (2) = origin_data (2) - origin_data (1)       = lat  - lon
derived_data (3) = origin_data (2) * origin_data (1)       = lat  * lon
...
```

So length_derived_data is equal to 3. For the first run with the increments for (lon , lat , par1 , par2) = (13 , 51 , 0.9 , 0.1) the following adjustments, transformations and associations will be performed:

	lon	lat	
<def>	0	0	
<adjustm>	S	S	
<incr>	13	51	
= value	13	51	
spatial_transform results in			
derived_data	51+13	51-13	51*13
=	64	38	663

Because of the defined association par1 : 2 is for

	par1	par2
<old_def>	1	2

<new_def>	38	= <old_def>
<adjustm>	M	A
<incr>	0.9	0.1
= value	34.2	2.1

and finally, for subroutine *simulation* input_data = (64 , 38 , 663, 34.2 , 2.1).

6.6.4 Recovering Association Problems under Simulation

Because of the computation of new default values for associated targets of the non-spatial subexperiment it is necessary to determine for unfeasible constellations how to compute target values from these new default values. This is done automatically during SPRINT-S performance according to the following rules to ensure continuation of the experiment. For interactive submission these actions are protocoled in an additional window. So it is possible to cancel an experiment. For the submission of an experiment to the LoadLeveler the additional window is suppressed. In both cases this protocol is appended to sprint.protocol.

experiment type	Warning or Error	adjustment type of assoc. target	<new_def>	recovery
spatial + behaviour	W	M	0	no
spatial + sensitivity	W	A , M	0	no
spatial + [perturbation | stochastic]	E	M for distribution U	0	<new_def> = 1 and lower boundary < upper boundary
spatial + [perturbation | stochastic]	E	M for distribution U	sign change	<new_def> = - <new_def>
spatial + [perturbation | stochastic]	E	A , M for distributions N, L	resulting standard deviation < 0	standard deviation = - standard deviation
spatial + [perturbation | stochastic]	E	A , M for distribution E	resulting mean < 0	mean = - mean
spatial + [perturbation | stochastic]	E	A , M for distributions N, L	resulting standard deviation = 0	standard deviation = 1
spatial + [perturbation | stochastic]	E	A , M for distribution E	resulting mean = 0	mean = 1

Tab. 3 Recovering association problems under simulation

6.6.5 Experiment Performance

Both the experiment performance and the storage of simulation results correspond with the appropriate strategy of each subexperiment. A sequence of subexperiments is performed in such a manner that the spatial subexperiment rules the complex experiment:

```
loop   spatial subexperiment
       loop   non-spatial subexperiment
              appropriate subexperiment sequence
       end loop
end loop
```

7 Linking a Model and Simulation Performance

7.1 Linking a Model to SPRINT-S

simulation, *prepare* and *spatial_transform* must be linked to SPRINT-S before running an experiment. To build the binary sprint.program, on a parallel node link the object file sprint.o and the SPRINT-S object archive libsprint.a from $SPRINTHOME together with the user modules for *simulation*.o, *prepare*.o, *spatial_transform*.o, the model under investigation and user object archives by

```
mpxlf  -o sprint.program  $SPRINTHOME/sprint.o  $SPRINTHOME/libsprint.a
       ......
```

and append to this link stream the above-mentioned user files. The name of the resulting binary must always be sprint.program. For SPRINT-S internally used subroutines, functions and common blocks, see Appendix I.

7.2 Environment Definition

Before starting a simulation experiment define and export the following UNIX environment on the parallel machine:

```
DISPLAY
MP_EUILIB
MP_INFO_LEVEL
MP_RMPOOL
SPRINTHOME
```

SPRINTSTATUS

For more information see Appendix D and E.

7.3 Interactive Experiment Submission

Start a simulation experiment directly at the parallel machine from the directory, where sprint.program resides by

$SPRINTHOME/*sprint.experiment*   <#proc>   <edf>   { <cmd> }

with

<#proc>	number of processors to be used: ≥ 3 maximal number depends on the number of available processors in parallel pool $MP_RMPOOL (see Appendix D)
<edf>	EDF
<cmd>	auxiliary command response file for performing *prepare* and followed *spatial_transform* in the case of a spatial experiment

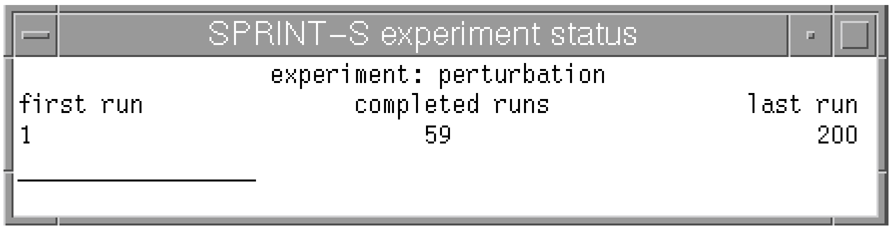
The CPU consumption for running an experiment scales linearly with the number of assigned simulation nodes. For SPRINTSTATUS = YES a pop-up window informs about the state of experiment performance. After announcing

experiment preparation
in progress

and

experiment performance
in progress

the currently finished run of the complete run sequence is output in the following way:



```
                  experiment: perturbation
first run               completed runs              last run
1                             59                         200
```

Fig. 5 Experiment status window

7.4 Experiment Submission to LoadLeveler

Submit a simulation experiment to the LoadLeveler by

$SPRINTHOME/*sprint.loadl* <#proc> <edf> { <cmd> }

with the same arguments as above. The user who submits the job to the LoadLeveler will be informed by an e-mail from the LoadLeveler when the job exits. For this submission type SPRINTSTATUS=NO is assumed.

7.5 Experiment Protocol and Output Redirection

Experiment performance is protocoled in the file sprint.protocol. This includes a statistics on CPU time for model simulation and node communication.

All information sent from *prepare*, *spatial_transform*, *simulation* or the simulation model to the terminal / standard output is redirected during a SPRINT-S experiment to sprint.protocol. Because of the parallel performance of the model, this information cannot normally be assigned to a special simulation run without a unique identifier.

7.6 Cleanup after Abnormal Experiment End

After an abnormal experiment end triggered by an error during performance of the simulation model or by cancellation of the experiment, the user should start from the directory in which the abnormal finished experiment was performed

$SPRINTHOME/*sprint.cleanup*

to delete all temporary files. For complex experiments this also cancels additional processes on the node the simulation was started from.

8 Result Processing

8.1 General Remarks

The general aim of result processing of SPRINT-S is to post-process simulation results and to output them in an experiment-specific manner, which allows further exploitation with additional software packages (e.g., visualization, statistics or geographic information systems' software).

One of the main features of SPRINT-S result processing is the possibility to determine from simulation results (the output data vector output_data or parts of it) extrema (minimum, maximum), moments (mean value, variance) and sums. This implies

- temporal aggregations, if parts of the simulation output represent state variables for different time steps
- class-specific aggregations, if parts of the simulation output represent state variables at a time step for a subclassification (e.g., biome distribution)

Result processing can be performed on any workstation by starting from the working directoryin which the simulation experiment was performed

$SPRINTHOME/*sprint.resproc*

Result processing always generates a plain ASCII output file sprint.res, where result values are separated from each other by horizontal tabulators. This file can be manipulated afterwards to serve as an input, e.g. for the above-mentioned packages.

During result processing the simulation experiment output file sprint.output is exploited. sprint.output

- is a direct access file of real*4 results
- where the sequence of records is explained for the different experiment types in the appropriate chapters under „Experiment performance“
- and the sequence of results for each record corresponds with the output vector output_data (1) ... output_data (length_output_data) of the subroutine *simulation*.

Output processing always starts for length_output_data > 1 with the determination of that part of the output vector from which results will be derived:

```
enter first and last position (max: <length_output_data>) in output data
      for result processing  (def: 1   <length_output_data>):
```

A singular interval (first position = last position) is possible. Because only one interval can be specified, this has to be kept in mind for the determination of the sequence of simulation results in *simulation* to enable the deviation of extrema and moments over connected intervals. Specification of aggregations over this interval is the next step:

```
enter aggregation  se/mi/ma/me/va/su  (def: se  ,  * for all other):
```

with

se	series	process each position
mi	minimum	determine minimum
ma	maximum	determine maximum
me	mean	determine mean value
va	variance	determine variance
su	sum	determine sum

at the selected interval. For singular intervals, se is active without any selection. In the following, selections for the current experiment type (see below) are performed.

The result output file sprint.res is headed with information on the processed experiment and on the result processing strategy. Result values are separated by a tabulator.

For lists of choices without a marked default value (def) the first list element always represents the default value in the following dialogues.

8.2 Behavioural and Spatial Analysis

Result processing can be performed on the basis of the specified combination or of the plain (independent of the combination in the EDF) simulation output file sprint.output:

```
result processing based on comb-line? (y/n):
```

For the latter case all runs are output sequentially for the determined aggregation at the selected interval, labeled by the appropriate run number. Result processing based on the combination within the experiment demands

- selection of target variables from the EDF body lines for which the results are to be presented and
- to fix all the other target variables, the selected target variable is combined with combinatorially.

The so-called level of freedom reflects this selection: It is one for aggregation se and two for aggregations unlike se and singular intervals:

```
fix target by increment position (def: 1) or select it to output by * for <target_list>
enter:
```

or

```
fix target by increment position (def: 1) for <target_list>
enter:
```

Implicit increment lists have to be resolved for position determination. Within such a dialogue sequence only a certain number of * can be entered, equal to the level of freedom. The dialogue is not performed if the level of freedom corresponds with the EDF combination in such a way that the level of freedom is fulfilled automatically.

If the selected <target_list> is a real list with targets combined in parallel, one target out of the <target_list> has to be selected by

```
select by its name the heading target (def: 1st target) from <target_list>
enter:
```

for labeling results. If the resulting values of the heading target are unsorted, they will be sorted in an ascending manner and the output will be organized in this way.

Example

With an EDF

```
E: behaviour
S p1 : 0 : 1 , 11 , 111
S p2 : 0 : 2 , 22 , 222
S p3 : 0 : 3 , 33 , 333
S p4 : 0 : 4 , 44 , 444
S p5 : 0 : 5 , -55 , 555
S p6 : 0 : 6 , 66 , 666
E: comb  p4,p1 * p5,p2 * p6,p3
```

result processing would be performed as follows

```
enter first and last position (max: 6) in output_data
        for result processing (def: 1    6):
enter aggregation se/mi/ma/me/va/su (def: se , * for all other):
result processing based on comb-line? (y/n):
fix target by increment position (def: 1) or select it to output by * for p4,p1
enter:
fix target by increment position (def: 1) or select it to output by * for p5,p2
enter: *
select by its name the heading target (def: 1st entry) from p5,p2
enter: p5
fix target by increment position (def: 1) for p6,p3
enter:
```

and result file sprint.res is for the model file example.f from $SPRINTHOME/example (see Appendix F) as

```
SPRINT-S    Vers. 2.1    Experiment: behaviour
output_data between 1 and 6
rows:           output_data
columns:        p5
fixed target:   p4 = 4
fixed target:   p6 = 6
```

```
se    -55     5    555
 1      1     1      1
 2     22     2    222
 3      3     3      3
 4      4     4      4
 5    -55     5    555
 6      6     6      6
```

Result processing dialogue as

```
enter first and last position (max: 6) in output_data
      for result processing (def: 1    6):
enter aggregation se/mi/ma/me/va/su (def: se , * for all other): *
result processing based on comb-line? (y/n):
fix target by increment position (def: 1) or select it to output by * for p4,p1
enter:
fix target by increment position (def: 1) or select it to output by * for p5,p2
enter: *
select by its name the heading target (def: 1st entry) from p5,p2
enter: p5
fix target by increment position (def: 1) or select it to output by * for p6,p3
enter: *
select by its name the heading target (def: 1st entry) from p6,p3
enter: p6
```

leads to the result file sprint.res as

```
SPRINT-S    Vers. 2.1    Experiment: behaviour
aggregation between output_data 1 and 6
rows:            p6
columns:         p5
fixed target:    p4 = 4

 mi           -55           5          555
  6           -55           1            1
 66           -55           1            1
666           -55           1            1
 ma           -55           5          555
  6            22           6          555
 66            66          66          555
666           666         666          666
 me           -55           5          555
  6       -3.1667         3.5     131.8333
 66       11.8333        18.5     146.8333
666      161.8333       168.5     296.8333
 va           -55           5          555
  6      702.1666         3.5   50618.1641
 66     1626.1666       687.5   46682.1641
666    80166.1641     76827.5   76622.1641
 su           -55           5          555
  6           -19          21          791
 66            71         111          881
666           971        1011         1781
```

8.3 Sensitivity Analysis

During a sensitivity experiment a nominal run and runs with positive and negative adjustments of the targets as defined in the EDF body lines are performed. During result processing sensitivity and symmetry functions for each selected position of output_data, each target and each increment will be derived from the outputs of the simulation runs. If a relative sensitivity function is undefined because a denominator is zero, then instead of the sensitivity function value the string „undef“ will be output.
If aggregations (extrema, moments, and sum) over the selected interval of output_data are to be output, they are derived from the calculated sensitivity function for all positions within this interval. For an aggregation of a relative sensitivity function with at least one undefined function value the whole aggregation will be undefined.

To navigate between targets and to select targets, increments and functions the following dialogue will be performed:

```
enter sensitivity function lin/squ/abs/rel1/rel2/sym (def: lin , * for all):
```

with sensitivity function <sens_function> and symmetry function

lin	linear sensitivity function
squ	squared sensitivity function
abs	absolute sensitivity function
rel1, rel2	relative sensitivity functions
sym	symmetry test

and two function values for positive and negative adjustments for all sensitivity functions.

To select the targets, the sensitivity and / or symmetry functions from which are to be output, the appropriate body line number has to be specified, followed by the increment position in the EDF terminator1-line for increment selection:

```
enter target position      (def: * for all) :
enter increment position   (def: * for all) :
```

Implicit increment lists have to be resolved for position determination.
In the case of multiple combinations the output sequence is determined by

```
enter output sequence for s_ensitivity functions,
                          t_argets and
                          i_ncrements              (def: sti) :
```

In the output file sprint.res sensitivity and symmetry functions are named as follows:

```
±<sens_function>_<target>(adjustment_type>:<incr>)
and / or
sym_<target>(adjustment_type>:<incr>)
```

Example

With an EDF

```
E: sensitivity
S: p1 : 0.5
S: p2 : 0
E: incr  0.01 , 0.05
```

result processing would be performed as follows

```
enter first and last position (max: 2) in output_data
        for result processing (def: 1    2):
enter aggregation se/mi/ma/me/va/su (def: se , * for all other):
enter sensitivity function lin/squ/abs/rel/sym (def: lin , * for all) : *
enter variable position (def: * for all) :
enter increment position (def: * for all) :
enter output sequence for        s_ensitivity functions,
                                 t_argets and
                                 i_ncrements              (def: sti) : tsi
```

and result file sprint.res is for the model file example.f from $SPRINTHOME/example (see Appendix F) as follows

```
SPRINT-S    Vers. 2.1    Experiment: sensitivity
output_data between 1 and 2

*                        1        2
+lin_par1(s:0.01)     -0.5        0          +lin_par2(s:0.01)      0        -1
-lin_par1(s:0.01)      0.5        0          -lin_par2(s:0.01)      0         1
+lin_par1(s:0.05)     -0.5        0          +lin_par2(s:0.05)      0        -1
-lin_par1(s:0.05)      0.5        0          -lin_par2(s:0.05)      0         1
+squ_par1(s:0.01)   0.0025        0          +squ_par2(s:0.01)      0      0.01
-squ_par1(s:0.01)   0.0025        0          -squ_par2(s:0.01)      0      0.01
+squ_par1(s:0.05)   0.0125        0          +squ_par2(s:0.05)      0      0.05
-squ_par1(s:0.05)   0.0125        0          -squ_par2(s:0.05)      0      0.05
+abs_par1(s:0.01)      0.5        0          +abs_par2(s:0.01)      0         1
-abs_par1(s:0.01)      0.5        0          -abs_par2(s:0.01)      0         1
+abs_par1(s:0.05)      0.5        0          +abs_par2(s:0.05)      0         1
-abs_par1(s:0.05)      0.5        0          -abs_par2(s:0.05)      0         1
+rel1_par1(s:0.01)      -1    undef          +rel1_par2(s:0.01)     0     undef
-rel1_par1(s:0.01)       1    undef          -rel1_par2(s:0.01)     0     undef
+rel1_par1(s:0.05)      -1    undef          +rel1_par2(s:0.05)     0     undef
-rel1_par1(s:0.05)       1    undef          -rel1_par2(s:0.05)     0     undef
+rel2_par1(s:0.01)    -0.5    undef          +rel2_par2(s:0.01)     0     undef
```

```
-rel2_par1(s:0.01)       0.5    undef          -rel2_par2(s:0.01)        0      undef
+rel2_par1(s:0.05)      -0.5    undef          +rel2_par2(s:0.05)        0      undef
-rel2_par1(s:0.05)       0.5    undef          -rel2_par2(s:0.05)        0      undef
sym_par1(s:0.01)           1        0          sym_par2(s:0.01)          0          2
sym_par1(s:0.05)           1        0          sym_par2(s:0.05)          0          2
```

8.4 Perturbation and Stochastic Analysis

During a perturbation or stochastic experiment, the ensemble of runs with perturbed targets is performed as well as a nominal run. Result processing addresses the task of information aggregation by presenting extrema, moments and boundaries of confidence intervals as well as heuristic distribution functions over the whole run ensemble. Experiment-specific dialogue is only performed for interval aggregation se and starts with

```
enter for heuristic distribution function number of classes (max: <nr_classes>)
        or 0 for no functions:
```

where <nr_classes> = (<nr_runs> - 1) / 4 is the number of equidistant, right opened intervals to be used as classes for heuristic distribution functions. Boundary classes are not treated specially. This dialogue is followed for non-zero inputs by determination of class boundaries for each item of the selected output_data interval or by determination of common boundaries for the complete interval:

```
determine class boundaries p_er item or (def.)
        or for the complete interval? (p/c):
```

Based on this class definition, heuristic distribution function can be plotted for the temporal aggregation se:

```
heuristic distribution to plot? (n/y):
```

Extrema, moments, confidence intervals and class frequencies over the whole run ensemble are named as follows:

dc	deterministic case
mi	minimum
ma	maximum
me	mean value
va	variance
sk	skewness
co1	distance from mean value for 99% confidence interval
co5	distance from mean value for 95% confidence interval
cl_1	class frequency in class number 1 (this is the class with the minimum value)
...	
cl_<nr_classes>	class frequency in class number <nr_classes>

(this is the class with the maximum value)

Interval-related aggregations are derived from the outputs of run ensemble related aggregations.

Example

With an EDF

```
E: perturbation
S par1 : 10 : u(0,2)
S par2 : 20 : n(0,2)
S par3 : 30 : l(0,2)
S par4 : 40 : e(1)
E: runs 500
```

result processing would be performed as follows:

```
enter first and last position (max: 4) in output_data
        for result processing (def: 1    4):
enter aggregation se/mi/ma/me/va/su (def: se , * for all other):
enter for heuristic distribution functions number of classes (max: 124)
        or 0 for no functions: 15
determine class boundaries p_er item (def) or for the c_omplete interval? (p/c):
heuristic distributions to plot? (n/y): y
```

and result file sprint.res is for the model file example.f from $SPRINTHOME/example (see Appendix F) as follows

```
SPRINT-S    Vers. 2.1    Experiment: perturbation
output_data between 1 and 4
499 runs, local heuristic variable distribution

output_data( 1 )  minimum: 0.9826660156E-02  width: 0.1324911714  max. frequ.: 41
  1 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  2 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  3 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  4 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  5 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  6 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  7 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  8 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  9 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 10 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 11 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 12 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 13 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 14 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 15 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

output_data( 2 )  minimum: -4.744464397  width: 0.7003035545  max. frequ.: 70
```

```
 1 xxxxx
 2 xxxxxxxxxxx
 3 xxxxxxxxxxxxxxxxxxxxx
 4 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 5 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 6 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 7 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 8 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 9 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
10 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
11 xxxxxxxxxxxxxxxxxxxxxxxxxx
12 xxxxxxxxxxxxxxxxxxxxxxx
13 xxxxxxxxxx
14 xxxxxxxx
15 x
```

(a.s.o.)

```
    *          1          2          3          4
   dc          1          0          0          4
   mi     0.0098    -4.7445     0.0033  0.8854E-3
   ma     1.9972     5.7601   275.9586     6.1777
   me     1.0157     0.0929     5.4242     1.0067
   va     0.3389     3.8317   417.8585     0.9645
   sk   696.2999   179.9835  0.1296E7    998.0846
  co1     0.0674     0.2266     2.3665     0.1137
  co5     0.0512     0.1722     1.7982     0.0864
 cl_1         34          6        474        169
 cl_2         27         11         12        117
 cl_3         38         22          4         72
 cl_4         34         35          3         36
 cl_5         29         57          2         36
 cl_6         41         67          1         21
 cl_7         29         70          0         20
 cl_8         31         62          0         14
 cl_9         33         57          1          6
cl_10         29         39          0          3
cl_11         31         28          0          0
cl_12         40         25          0          2
cl_13         27         10          0          1
cl_14         39          8          1          0
cl_15         37          2          1          2
```

8.5 Complex Experiments

A special result processing approach for complex experiments is the selection of a special super-run out of the spatial subexperiment:

```
fix a single super-run from the spatial subexperiment:
fix target by increment position (def: 1) for <target_list>
enter:
```

...
perform now result processing for the non-spatial subexperiment:

Example

For the EDF **complex2**

```
E: spatial + behaviour
assoc  par1 : 2
E: files spatial.edf + behav.edf
```

with spatial.edf and behav.edf from the chapter on complex experiments

result processing would be performed as follows:

```
enter first and last position (max: 23) in output_data
        for result processing (def: 1    23):
enter aggregation se/mi/ma/me/va/su (def: se , * for all other):
fix a single super-run from the spatial subexperiment:
fix target by increment position (def: 1) for lon: 6
fix target by increment position (def: 1 ) for lat: 1
perform now result processing for the non-spatial subexperiment:
... (dialogue for the non-spatial subexperiment)
```

and result file sprint.res is as follows

```
SPRINT-S    Vers. 2.1    Experiment: spatial + behaviour
output_data between 1 and 23
fixed spatial super-run:
fixed target: lon = 13.5
fixed target: lat  = 53
... (output for the non-spatial subexperiment)
```

8.6 Experiment Storage

To save simulation results of an experiment for later result postprocessing, rename or restore

- sprint.output
- sprint.scenario
- sprint.protocol
- sprint.input (only for behavioural and spatial experiments)
- sprint.input_c (only for complex experiments)
- sprint.scenario_c (only for complex experiments)

and assign the original names before starting result processing from the directory, in which these files are located.

8.7 Browsing Simulation Results

It may sometimes be useful to browse simulation results stored in sprint.res without taking into account experiment-specific-result performance, as coded in the control files sprint.scenario and sprint.scenario_c (the latter only for complex experiments). For this purpose SPRINT-S supplies the user with

$SPRINTHOME/*sprint.browse*

where the user has to enter on request

- the length length_output_data of the output data vector output_data from subroutine *simulation*
- first and last position in output data vector output_data for browsing
- first and last record of sprint.output for browsing
- for complex experiments the number of records per super-run
- for complex experiments the run numbers to output per super-run

Keep in mind that for perturbation and stochastic analysis the number of stored records is equal to the number of runs to be performed plus 8. Especially for complex experiments the selection of run numbers to be output per super-run is a valuable approach for result interpretation.

$SPRINTHOME/*sprint.browse* has to be started from the directory from which the SPRINT-S experiment was performed. It always generates a plain ASCII output file sprint.brw, where values are separated by horizontal tabulators.

8.8 Result File Filters

Result filters usually can be applied to plain ASCII result output files sprint.res and sprint.brw or derived files to transform results to be used as input to other packages. The directory $SPRINTHOME/filters is a repository for such filters. For the available filters see Appendix C.

9 References

Flechsig, M., Wenzel, V., Erhard, M. (1994) Simulation based regional models - concept, desgin and application. Ecol. Mod. 75/76, 601-608

IBM (1994) AIX parallel environment - parallel programming subroutine reference. IBM Document No. SH26-7228-01

King, A.W. (1990) Translating models across scales in the landscape. In: Turner, M.G. & Gardner, R.H. (eds): Quantitative methods in landscape ecology. Ecological Studies Vol. 82, Springer, New York, pp. 479-517

Wenzel, V., Kücken, M., Flechsig, M. (1995) MOSES - Modellierung und Simulation ökologischer Systeme. PIK-Report No. 13, PIK Potsdam

Wenzel, V., Matthäus, E., Flechsig, M. (1990) One decade of SONCHES. Syst. Anal. Mod. and Sim. 7, 411-428

Appendix A
Subroutines and Functions for Model Coupling

The user can / must use the following SPRINT-S subroutines and functions for coupling his / her model to SPRINT-S:

name and explanation	**arguments or function value**	**input / output and type**	**meaning**	**to be used in or usable in**	**see**
subroutine ***broadcast***	field_to broadcast	input integer*4 or real*4	field to broadcast from *prepare* to *simulation* or the simulation model	usable in: *prepare*	4.1
broadcast information to simulation model	length_field_to_ broadcast	input real*4	length of field_to_broadcast		
function ***iget_exp_type*** get experiment type	iget_exp_type	output real*4		usable in: *prepare* and *simulation*	4.1
subroutine ***put_length_ derived_data*** put length of output vector for *spatial_transform*	length_derived_ target	input integer*4	length of output vector derived_data from *spatial_transform*	to be used in: *prepare* for spatial analysis	4.1
subroutine ***put_length_ output_data*** put length of output vector for *simulation*	length_output_ data	input integer*4	length of output vector output_data from *simulation*	to be used in: *prepare*	4.1, 6.5
function ***stochastic*** generate a random number	stochastic	output real*4		usable in: *simulation* for stochastic analysis	6.4
subroutine ***synchro*** synchronize *broadcast*				usable in: *prepare*	4.1

Tab. 4 User subroutines and functions

The user has to define the following subroutines in order to couple a model to SPRINT-S:

name and explanation	**arguments**	**input / output and type**	**meaning**	**see**
subroutine ***prepare*** prepare experim.	at_comm_node	input logical*4	.true. if node is the communication node .false. else	4.1
subroutine ***simulation*** wrap simulation model and perform it for a single run	input_data	input real*4	target vector, derived from the EDF and transformed for spatial experi-ments by subr. *spatial_transform*	4.2
	length_input_data	input integer*4	length of input_data. Determined by the EDF or for spatial experiments in *prepare* by subroutine *put_length_derived_data*	
	output_data	output real*4	output vector of simulation results	
	length_output_data	input integer*4	length of output_data. Fixed in *prepare* by subroutine *put_length_output_data*	
subroutine ***spatial_transform*** transform target vector for spatial experiments	origin_data	input real*4	original targets as defined in the EDF. Corresponds with sequence of EDF body lines	6.5
	length_origin_data	input integer*4	length of origin_data. It is determined by the number of EDF body lines	
	derived_data	output real*4	result of tranformation *in spatial_transform*. It will be used as argument input_data in subroutine *simulation*	
	length_derived_data	input integer*4	length of derived_data. It is determined within prepare by the subroutine *put_length_derived_data*	

Tab. 5 Subroutines to be defined by the user

SPRINT-S function iget_exp_type returns the following values for experiments:

experiment type	integer*4 *iget_exp_type* ( )
spatial	1
behaviour	2
sensitivity	3
perturbation	4
stochastic	5
spatial + behaviour	12
spatial + sensitivity	13
spatial + perturbation	14
spatial + stochastic	15

Tab. 6 Function values of iget_exp_type

Appendix B
Scripts for Model Handling

The following SPRINT-S scripts are available from $SPRINTHOME at the operating system prompt:

script	explanation	on (sp) or on (ws)	arguments	
sprint.experiment	submit an experiment interactively	sp	#proc edf cmd	number of processors EDF optional command file
sprint.loadl	submit an experiment to the LoadLeveler	sp	#proc edf cmd	number of processors EDF optional command file
sprint.moses	perform an experiment with a MOSES model inter-actively	sp	#proc edf model ext cmd	number of processors EDF MOSES model name MOSES model extension optional command file
sprint.resproc	result processing	ws	---	
sprint.browse	browse result file sprint.output	ws	---	
sprint.cleanup	clean current directory after abnormal experiment end	ws	---	

sp stands for: run this script only on sp
ws stands for: run this script on any workstation

Tab. 7 Scripts for model handling

Usage of a script with arguments can be acquired by entering the script name without arguments, e.g., sprint.experiment.

Appendix C
Result Output Filters

The following filters for transforming result output files sprint.res, sprint.brw or derived files are currently available from $SPRINTHOME/filters.

The first argument has to be always the input file to be transformed, additional arguments are listed below. All filters output to standard output. All filters can be run on any workstation. For more filters check $SPRINTHOME/filters/README.

filter	application	additional arguments
2transpose	transpose result output files heading records up to first line with a tabulator: string „row“ ----> string „column“ string „column“ ----> string „row“ up from first line with a tabulator: transpose items, separated by tabulators	----
2arcinfo	transforms to ARC/INFO INFO import format: tab ----> newline newline ----> blank line	----
2excel	transforms to EXCEL import format: point ----> comma	----

Tab. 8 Result output filters

Appendix D
Operating System Environment

The following UNIX environment variables have to be set and exported before running SPRINT-S:

environment	value	description
for all parts of SPRINT-S:		
SPRINTHOME		the SPRINT-S home directory: see Appendix E
for experiment performance		
SPRINTSTATUS	 YES NO	experiment status in an extra window useful only for CPU-intensive experiments with CPU-time > 3 sec per simulation run and interactive submission otherwise
DISPLAY		output terminal / only for SPRINTSTATUS=YES
MP_INFOLEVEL	 0 1 2 3 4	message reporting to sprint.protocol set MP_INFOLEVEL > 1: if the error occurs: ERROR 0031-214 pmd chd <working dir.> (parallel operating system environment bug) set MP_INFOLEVEL = 1: otherwise errors 0 and warnings (default) 1 and informational warnings 2 and high level diagnostic messages 3 and low level diagnostic messages
MP_RMPOOL		number of the parallel system pool, from which the experiment is to be performed (see Appendix E)
MP_EUILIB	 ip us	CSS library implementation to use Internet protocol for communication among processors drive high performance switch directly from the parallel tasks. Fastest communication among nodes, but only one parallel job can run in this mode.
set automatically by SPRINT-S for experiment performance		
MP_PROCS	<#proc>	number of nodes to be used from experiment command line
MP_EUIDEVICE	css0	adapter set to be used for MP_EUILIB=ip
MP_RESD	YES	whether or not to use parallel system resource manager
MP_HOSTFILE	NULL	host list file name
SPRINTSTATUS	NO	experiment status in an extra window for experiment submission to LoadLeveler

Tab. 9 Operating system environment

For detailed information on the parallel environment see IBM's AIX Parallel Environment documentation.

Appendix E
Current Settings, Home Directory, and Restrictions

E.1 Current settings

environmental variable	current setting
SPRINTHOME	/usr/local/sprint-s
MP_RMPOOL	should be set equal to 2 parallel pool 2 has 46 nodes

Tab. 10 Current settings

E.2 Home Directory

The SPRINT-S home directory $SPRINTHOME has the following structure:

directory	contents
$SPRINTHOME	binaries and scripts
$SPRINHOME/filters	result output files filters (binaries and scripts)
$SPRINTHOME/example	example model example.f and EDF examples
$SPRINTHOME/doc	documentation

Tab. 11 Home directory structure

E.3 Current Restrictions and Workarounds

task and experiment types	limitation	workaround
experiment performance: all experiment types	number of EDF body lines is limited to 100	----
result processing: complex experiments	result processing only available when fixing a super-run	use *sprint.browse*

Tab. 12 Current limitations and workarounds

Appendix F
Examples

To clarify the interplay between SPRINT-S, *simulation*, *prepare*, *spatial_transform,* the simulation model and the EDF the following problem definition is considered:

F.1 Problem Definition

Problem 1

A behavioural analysis is to be performed for a model veg_dyn of vegetation dynamics at geographic coordinates longitude lon = 13° and latitude lat = 52° for two model parameters.

Problem 2

A regional application of the laterally uncoupled model veg_dyn within a box bounded by (12° , 51.5°) and (14° , 53.5°) with a regular stepwidth of 0.05° is to be performed.

It is assumed that the model veg_dyn

- maps dynamics for one patch / grid element
- should be performed under common meteorological driving forces over the whole region
- is dependent on 22 patch- / grid element-specific values (longitude, latitude and parameters and / or initial values, e.g., soil, vegetation types, vegetation distribution, ...), that are stored in an external data base / data file with an identificator (lon,lat) for each patch / grid element
- has 100 result values per model run

The example describes an approach where the user subroutines *prepare* and *simulation* are defined for all of the above problems, using the SPRINT-S function *iget_exp_type* for distinction. The used subroutines meteo_data and site_data are user-defined subprograms.

F.2 EDF Definition

Define an EDF behav.patch for **Problem 1**

```
    E: behaviour
    M vegdynp3 : 3 : 0.9 (0.02) 1.1
    M vegdynp6 : 6 : 0.9 (0.05) 1.1
    E: comb default
```

resulting in 11 * 5 = 55 runs to inspect the parameter subspace
(vegdynp3 , vegdynp6) of veg_dyn.

Define an EDF spatial.region for **Problem 2**

```
    E: spatial
    S lon : 0 : 12(.05)14
    S lat  : 0 : 51.5 (.05) 53.5
    E: comb default
```

resulting in 41 * 41 = 1681 runs for model performances on a regular 0.05° x 0.05° geometry.

F.3 Definition of Subroutine *prepare*

```
      subroutine prepare ( at_comm_node )
      logical*4 at_comm_node
      common /veg_dyn_meteo/ prec (3650) , temp (3650)
      real*4 prec , temp
      common /veg_dyn_params/ veg_dyn_lon , veg_dyn_lat , veg_dyn_param (20)
      real*4 veg_dyn_lon, veg_dyn_lat, veg_dyn_param

c     set length_output_data:
      call put_length_output_data ( 100 )
c     for spatial experiments (Problem 2)
c     set length_derived_data = length_input_data,
c     for all other experiments length_input_data is determined
c     automatically by exploiting the EDF
      if ( iget_exp_type ( ) .eq. 1 ) call put_length_derived_data ( 22 )
c     read daily meteorological data at the communication node
c     for 10 years, valid for all patches/grid elements
c     to veg_dyn common /veg_dyn_meteo/
c     avoiding the data to be read each time in veg_dyn:
      if ( at_comm_node ) then
            call meteo_data ( ‘open’ )
            call meteo_data ( ‘read’ , prec )
            call meteo_data ( ‘read’ , temp )
            call meteo_data ( ‘close’ )
      endif
c     broadcast commons parallely to all simulation nodes, using broadcast
c     imbed broadcasts within synchro:
      call synchro
      call broadcast ( prec , 3650 )
      call broadcast ( temp , 3650 )
      call synchro
c     connect to the external site-specific database at the communication
c     node:
      if ( at_comm_node ) call site_data ( ‘open’ )
c     only for the behavioural analysis (Problem 1) read site-specific
c     information for (13°,52°) at the communication node,
c     broadcast it to all simulation nodes
c     and close external site-specific database:
      if ( at_comm_node ) then
            if ( iget_exp_type ( ) .eq. 2 ) then
                  veg_dyn_lon = 13.
```

```
              veg_dyn_lat = 52.
              call site_data( ‘read’ , veg_dyn_lon , veg_dyn_lat ,
     #                          veg_dyn_param , 20 )
              call synchro
              call broadcast ( veg_dyn_lon , 22)
              call synchro
              call site_data ( ‘close’ )
        endif
      endif
      return
      end
```

F.4 Definition of Subroutine *spatial_transform*

```
      subroutine spatial_transform ( origin_data  , length_origin_data ,
     #                               derived_data , length_derived_data )
c     spatial_transform is only supplied for the spatial experiment
c     (Problem 2)
      integer*4 length_origin_data , length_derived_data
      real*4 origin_data (length_origin_data)
      real*4 derived_data (length_derived_data)

c     as indicated by the two EDFs  origin_data (1) = lon
c                                   origin_data (2) = lat
c     get now from the external site-related data base dependent on
c     lon and lat the site-specific 20 data and copy it directly
c     to derived_data
      call site_data ( ‘read’ , origin_data (1) , origin_data (2) ,
                       derived_data (3) , 20 )
c     copy lon and lat to the head of the derived_data
      derived_data (1) = origin_data (1)
      derived_data (2) = origin_data (2)
c     so, length_derived_data = 22, as set within prepare
      return
      end
```

F.5 Definition of Subroutine *simulation*

```
      subroutine simulation ( input_data  , length_input_data ,
     #                        output_data , length_output_data )
      integer*4 length_input_data , length_output_data
      real*4 input_data (length_input_data)
      real*4 output_data (length_output_data)
   c  veg_dyn common of internal parameters
      common /veg_dyn_params/ veg_dyn_lon , veg_dyn_lat , veg_dyn_param (20)
   c  veg_dyn_common of internal model results
      common /veg_dyn_outputs/ veg_dyn_output (200)

   c  for the spatial experiment (Problem 2):
   c  map input_data (1 ... 22)
   c  on the appropriate parameters and initial values of veg_dyn:
      if ( iget_exp_type ( ) .eq. 1 ) then
            veg_dyn_lon = input_data (1)
            veg_dyn_lat = input_data (2)
            do i = 3 , length_input_data
                  veg_dyn_param (i) = input_data (2+i)
            enddo
      else
c     for experiment type Behavioural Analysis (Problem 1):
c     map input_data (1 ... 2) on veg_dyn parameters 3 and 6
            veg_dyn_param (3) = input_data (1)
            veg_dyn_param (6) = input_data (2)
      endif
c     run veg_dyn; within veg_dyn there exists the common
c     common /veg_dyn_meteo/ prec (3650) , temp (3650)
c     where meteorological data are used without changing it.
      call veg_dyn
c     store simulation results to output_data:
      do i = 1 , length_output_data
            output_data (i) = veg_dyn_output (i)
      enddo
      return
      end
```

F.6 $SPRINTHOME/example

To address once again SPRINT-S management of data in connection with *prepare*, *simulation* and *spatial_transform*, the directory $SPRINTHOME/example contains another example for coupling a model to SPRINT-S. This example with the source file example.f, generally speaking copies input data to output data, without performing any user model. Additionally, appropriate EDFs are stored as edf*.<experiment_type>. For more information see example.f. To run SPRINT-S with this model, copy at least sprint.program to a directory with write access.

Appendix G Differences for MOSES Models

For MOSES models (Wenzel et al., 1995) the following differences exist with respect to all other models:

- ***prepare*** and ***simulation*** are provided by SPRINT-S for each model.

- In the absence of sprint.program the linker is performed automatically before starting an experiment.

- EDF body line

  <target_type> <adjustment_type> <target> { : <exp_spec_infos> }

  with

<target_type>	[P | S | T]
	where
	P for parameter
	S for initial value

T	for table

For <target_type> = T independently on the value of <adjustment_type> <adjustment_type> = S is active. Then increments are table names.
The <target> item is case sensitive.

- To perform an experiment interactively, start

  $SPRINTHOME/*sprint.moses* <#proc> <model> <ext> <edf> {<cmd>}

  with

<#proc>	number of processors to be used: ≥ 3 maximal number depends on the number of available processors in parallel pool $MP_RMPOOL (see Appendix D)
<edf>	EDF with file extension <model>
<model>	model name
<ext>	model extension
<cmd>	optional command file

- The following non-public subroutines and functions are additionally (see Appendix I) used for that part of SPRINT-S which is to be linked to the MOSES model to be performed:

  iadr , iadsb3 , infno , ipos , istart , length_res_run , numb

- The following named common blocks are additionally (see Appendix I) used in SPRINT-S for that part of SPRINT-S which is to be linked to the MOSES simulation model to be performed:

  ab , lun , no1

Appendix H
Derived Files

The following files are generated by SPRINT-S in the working directory from which the simulation was started:

file name	contents
sprint.program	parallel simulation binary to be linked by the user
sprint.scenario	control file, derived from the EDF For complex experiments this is the control file of the non-spatial subexperiment
sprint.scenario_c	only for complex experiments: control file of the spatial subexperiment
sprint.input	ASCII increment file for Behavioural and Spatial Analysis. For the complex experiments this is the input file of the non-spatial subexperiment
sprint.input_c	only for complex experiments: ASCII increment file of the spatial subexperiment
sprint.output	direct access binary file of simulation results. Each record contains the results of one simulation run
sprint.protocol	preparation and simulation ASCII protocol file of the experiment
sprint.loadl_*	additional protocol files from the LoadLeveler for experiment submission to the LoadLeveler
sprint.res	ASCII output file after result processing with $SPRINTHOME/sprint.resproc
sprint.brw	ASCII output file after result browsing with $SPRINTHOME/sprint.browse
sprint.tmp*	auxiliary files, only for $SPRINTSTATUS=YES

Tab. 13 Derived files during an experiment

Appendix I
Subroutines, Functions, Common Blocks and Logical Unit Numbers

I.1 Subroutines and Functions Used

The following SPRINT-S subroutines and functions are available to users (see Appendix A):

broadcast , iget_exp_type , put_length_derived_data , put_length_output_data , stochastic , synchro

The following non-public subroutines and functions are used for that part of SPRINT-S which is to be linked to the simulation model to be performed:

acc , allocate , b_blank , b_copy , b_zero , c_copy , check , complex_exp , c_opt , char1 , cumul , deallocate , default_value , distr_* , filesize , icomp, i_copy , i_deco , i_enco , iget_ele , igetarg , iget_inp_data , incr_adjust_* , incr_field , i_opt , iput_res_data , i_read , i_zero , kill_blank , lencha , low_case , mp_* , random_init , random_number , readreal , r_copy , r_deco , r_enco , r_opt , r_read , r_zero , scen_simu , scenario , send_input , sleep , sprint_work , sprint (main program) , status , t , task_simu_* , spatial_transform (dummy module) , veri

I.2 Common Blocks Used

The following named common blocks are used in SPRINT-S for that part of SPRINT-S which is to be linked to the simulation model to be performed:

decoenco , ein , equiv , exp_descr1 , exp_descr2 , pointers , sprint1 , sprint2 , ... , sprint9 , sprint10

I.3 Logical Unit Numbers Used

The logical unit numbers (LUNS) 100 to 105 are used for file input and output for that part of SPRINT-S which is to be linked to the simulation model to be performed.