PIANOS design document

Group Linja

Helsinki 29th June 2005

Software Engineering Project

UNIVERSITY OF HELSINKI
Department of Computer Science

Course

581260 Software Engineering Project (6 cr)

Project Group

Joonas Kukkonen
Anni Kotilainen
Marja Hassinen
Eemil Lagerspetz

Client

Marko Salmenkivi

Project Masters

Juha Taina
Vesa Vainio (Instructor)

Homepage

`http://www.cs.helsinki.fi/group/linja`

Contents

1 Preface

1.1 Version history

Version	Date	Modifications
0.2	15.06.2005	The document template

2 Glossary

2.1 Probabilistic inference

Random variable (synonym: stochastic variable): Function which combines events with their probabilities. A numeric variable related to a random phenomenon (for example throwing a dice). The value of the variable is determined if the result of the phenomenon is known, otherwise only the probabilities of different values are known. [Tode04]

Discrete random variable: Random variable with a discrete range.

Point probability function: Function $f : R \rightarrow R$ related to discrete random variable X so that $\forall x \in R : f(x) = P\{X = x\} =$ the probability that the value of X is x. [Tode04]

Density function: $f(x)$ is a density function iff

1. $f \geq 0$
2. f is integrable in R and $\int_{-\infty}^{\infty} f(x)dx = 1$.

[Tode04]

Cumulative distribution function: The function $F : R \rightarrow R$ is the cumulative distribution function associated with random variable X iff $F(x) = P\{X \leq x\} =$ the probability that X is less than or equal to x. [Tode04]

Continuous distribution: A random variable has a continuous distribution as its density function f if $\forall a, b \in R : P\{a \leq X \leq b\} = \int_a^b f(x)dx$ [Tode04]

Joint distribution: If X and Y are random variables, their joint distribution describes how probable all the possible combinations of X and Y are.

Independence: Events A and B are independent, if the probability of B is independent on whether A has happened or not. (For example if A = "it rains", B = "when I throw a dice the result is 6" and C="when I throw a dice the result is even" then A and B are independent but B and C are not.) [Tode04]

Conditional probability: If X and Y are random variables, the conditional probability $P\{X = x|Y = y\}$ means the probability that the value of X is x if it is assumed that the value of Y is y.

Bayes's rule: $P\{X = x|Y = y\} = \frac{P\{Y=y|X=x\} \cdot P\{X=x\}}{P\{Y=y\}}$. The rule is obtained from the observation that $P\{Y = y|X = x\} \cdot P\{X = x\} = P\{X = x \text{ AND } Y = y\}$

Chaining: Using the Bayes's rule many times consecutively.

Bayesian model: A model which connects dependent random variables to each other by defining dependencies and conditional probabilities. The model defines the joint distribution of the variables.

Likelihood function: $P\{data|explanation\}$ is called the likelihood function because it defines how likely it is to get such a data if the real conditions are known. (For example if we know that a bird resides at some area, then how likely it is to get the observations we have already got.)

Markov random field: A random field that exhibits the Markovian property:
$\pi(X_i = x_i | X_j = x_j, i \neq j) = \pi(X_i = x_i | \delta_i)$, Where δ_i is the set of neighbours for X_i. That is, only the adjacent units affect the conditional probability. [Rand]

Discrete distributions:

- **Uniform distribution**
- **Binomial distribution**
- **Geometric distribution**
- **Poisson distribution**

Continuous distributions:

- **Uniform distribution**
- **Normal distribution**
- **Multinomial distribution**
- **Exponential distribution**
- **Binormal distribution**
- **Lognormal distribution**
- **Beta distribution**
- **Gamma distribution**
- **Dirichlet distribution**

Descriptions of these distributions can be found in [Math05].

Functional dependence: A condition between two variables X and Y so that the value of X determines the value of Y unambiguously.

Stochastic dependence: A condition between two variables X and Y so that the value of X doesn't determine the value of Y but influences the probabilities of the possible values.

Spatial dependence: A special case of stochastic dependence where the dependence is related to some spatial structure. (For example towns that are adjacent to each other influence to each other.) The spatially dependent variables form a Markov random field.

Prior distribution: The prior distribution of the parameters describes their assumed joint probability distribution before inferences based on the data are made.

Posterior distribution: The posterior distribution of the parameters describes their joint probability distribution after inferences based on the data are made.

Marginal distribution: The prior or posterior distribution concerning only one parameter.

2.2 Models and related concepts

Model: Means: Bayesian model

Variable: A variable is an entity in the model that can have an assigned value from its range of values. Variables and their dependencies form the base of the problem that the software is developed to solve.

Parameter: A parameter is a variable whose value is not defined by data.

Adjacency matrix: The adjacency matrix of a simple graph is a matrix with rows and columns labeled by graph vertices, with a 1 or 0 in position ij according to whether i and j are adjacent or not.

Floating point number: A computer representation of a real number with finite precision.

2.3 Metropolis-Hastings algorithm

Iteration: A single round of the algorithm when all the parameters have been updated once.

Burn-in-iterations: The iterations that are run before any output is produced.

Thinning factor: The thinning factor t means that every t^{th} iteration value is used in the output and the rest are discarded.

Block: A set of parameters which are defined to be updated together. That is, the proposals are generated to all of them and the acceptance of all the proposals is decided at the same time.

Update: Proposing a value to a parameter and then accepting it (the value changes) or discarding it (the value remains the same).

Proposal strategy: The proposal strategy defines how the next proposed value is generated. Possible choises are

1. Fixed proposal strategy: The next proposed values for a parameter is taken from its proposal distribution.

2. Random walk: The next proposed value for a parameter is created by adding a value taken from the proposal distribution to the current value of the parameter.

Proposal distribution:

1. The distribution from which the next proposed value for a parameter is chosen (when using the “Fixed proposal distribution” proposal strategy).

2. The distribution that is used in generating proposed values for a parameter by adding a value taken from the distribution to the parameter’s current value (when using the “Random walk” proposal strategy).

Update strategy: The update strategy describes which variables belong to the same block. (See: Block) The update strategy also includes information about whether the blocks are considered for updates in sequential order, whether the next block to update is chosen at random or whether the block to update is chosen based on the block weights.

Convergence: The phenomenon that during the simulation the parameter values get closer to the posterior distribution. The speed of the convergence depends on the initial values and other simulation parameters.

3 Architecture

This section will contain diagrams of the division of the software into the generator and the executable program the generator produces.

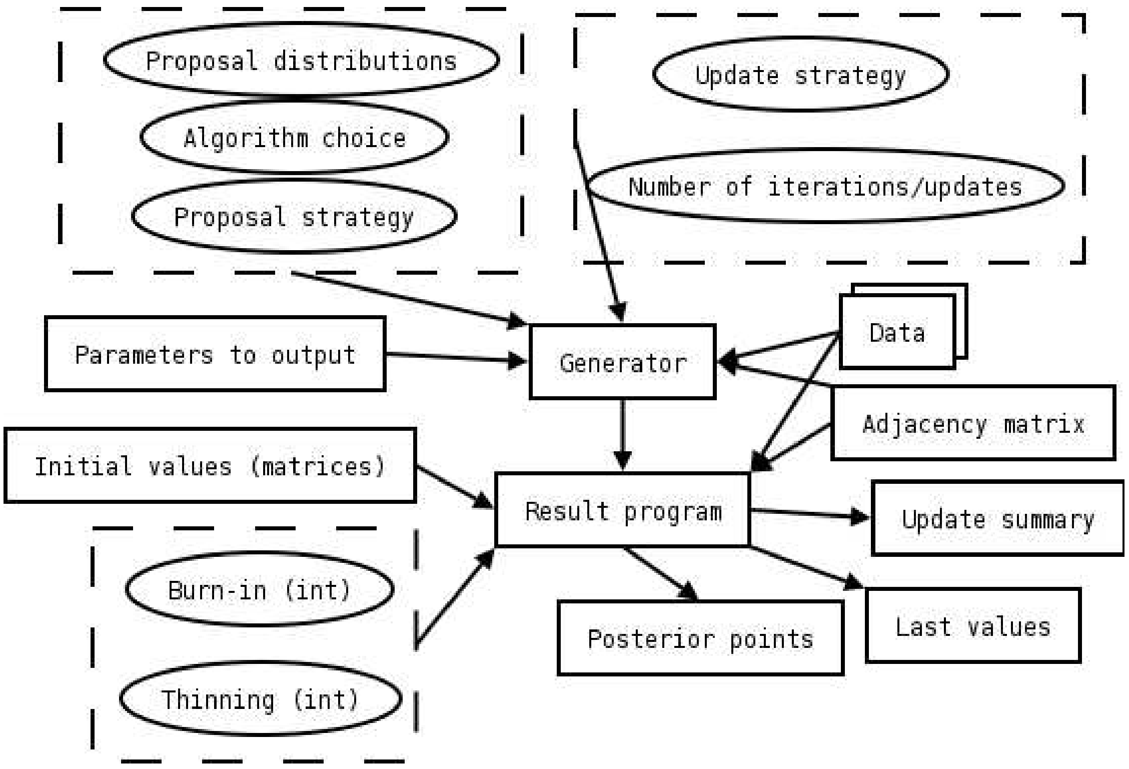

Figure 1: A diagram of the software input/output files and data flow.

Figure 1 represents the generator and the result program that it produces, and how data is used by them. Every rectangle and dotted line rectangle is a file (except the generator and the result program). An arrow means that data from the component at the source of the arrow is used by the component that the arrow leads to.

4 The format of input files

This section describes the format of the input files, that is the model description and the files for the simulation technical parameters.

4.1 Model description language

A model description consists of variable and entity definitions. It can also include comment lines, any line beginning with '#' is considered a comment. The order of the lines or capitalization make no difference.

Variables can be either integers or floating point numbers. This is expressed with a 'int' or 'float' in front of their names.

Examples:

```
float alpha ~ Gamma(0.1, 1.0)
```

Alpha is a stochastic variable with a Gamma distribution. The distribution's parameters are 0.1 and 1.0.

```
int beta = alpha + delta^2
```

Beta is a functional variable. It's value is alpha + delta^2.

```
day, "day.txt" {
    x(10)
}
```

Day is an entity. The value of x_i is found in the file day.txt on row i, col 10.

4.1.1 Formal syntax

An explanation of the notation used can be found at

http://www.python.org/dev/doc/devel/ref/notation.html

model: (variable | entity | comment)*

variable: stochastic_variable | functional_variable

stochastic_variable: type ' ' name '~' distribution

functional_variable: type ' ' name '=' equation

type: 'int' | 'float'

entity: name [‘,’ file [‘,’ file] [‘, combines(’ name ‘,’ name ‘)’]] ‘{’ variable* ‘}’

comment: ‘#’ .*

name: letter*

distribution:

equation:

file: letter* ‘.txt’

letter: ‘a’...‘z’

4.2 Simulation parameters as input

4.2.1 The burn-in length and the thinning factor

The burn-in length and the thinning factor are both integers. They must be given in a single file, given like in the example below:

```
? burn-in
1000

? thinning
4
```

4.2.2 Blocking and the algorithm used

A block is defined by listing parameters which belong to it. The format is

```
(parameter1, parameter2)
(parameter3, parameter4, parameter5)
etc.
```

If the user wants to define that the Gibbs sampling algorithm must be used when updating the block, the key word “Gibbs” is entered after the block’s definition. For example:

```
(alpha_1, beta_3, gamma) Gibbs
```

The default algorithm used is the Metropolis-Hastings algorithm.

If the block contains only one parameter, it doesn’t have to be defined unless the user wants to use the Gibbs sampling algorithm.

4.2.3 Initial values

The user must provide initial values for all parameters. The initial values are given in a single file.

The first line of the file must be:

```
?? initial values
```

The format of the file is:

1) Legend, which can be one of the following:

```
? parametername

? parametername begin:end

? parametername begin:end begin2:end2
```

2) The initial values in an array or a matrix corresponding to the legend.

The file may contain several consecutive legend - values - pairs.

An example:

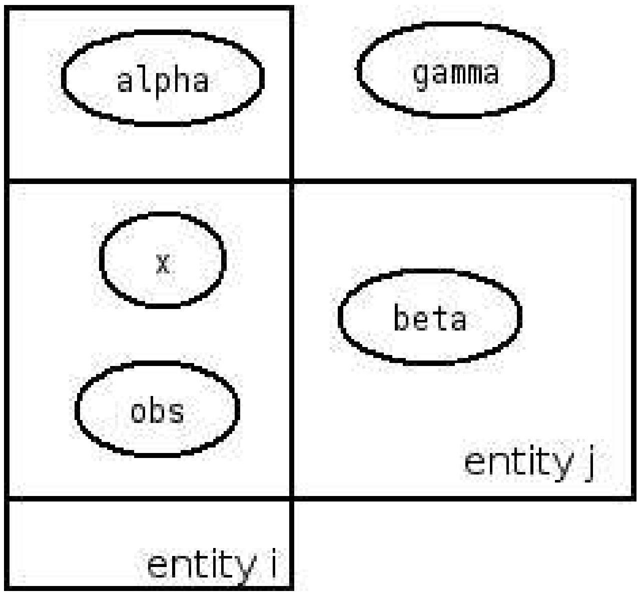

Figure 2: A model with two intersecting entities

```
? gamma
4.0
```

The legend means that we are giving an initial value to a single variable on the line after the legend.

```
? alpha 1:5
3.2
0.3
3.1
5.3
2.9
```

The legend describes that we are giving initial values to $alpha_1, alpha_2, alpha_3, alpha_4$ and $alpha_5$ in an array on the lines following the legend line. After setting the initial values, we have $alpha_1 = 3.2, alpha_2 = 0.3$ etc.

```
? beta 1:3
5.4
2.1
1.0
```

The legend means that we are giving initial values to $beta_1, beta_2$ and $beta_3$ in an array on the lines after the legend. After setting the initial values, we have $beta_1 = 5.4, beta_2 = 2.1$ etc.

```
? x 1:5 1:3
1.0 3.2 5.3
1.2 5.2 3.1
3.8 4.1 8.0
4.4 2.7 3.2
5.6 0.4 1.2
```

The legend describes that we are giving initial values to $x_{1,1}, x_{1,2}, x_{1,3}, x_{2,1}, ...x_{5,3}$. The values are in a matrix form after the legend line. The first index corresponds to the vertical dimension and the second index corresponds to the horizontal dimension of the matrix. After setting the initial values, we have $x_{3,2} = 4.1$ for example.

If some instances of, say, the variable obs are missing from the data, the user must provide initial values for them.

```
? obs 4:4 1:1
4.3
```

This definition states that the initial value of $obs_{4,1}$ is 4.3.

4.2.4 Proposal strategies and proposal distributions

The format of the proposal strategies and distributions input file is similar to the initial values input file.

The first line of the file must be:

```
?? proposal distributions
```

An example of giving the proposal distributions:

```
? alpha all
Poisson(3)
```

This expression states that the parameters $alpha_1, ..., alpha_5$ have the same proposal distribution Poisson(3) and the fixed proposal strategy is used as default.

```
? x all
Norm(0, 0.1) RW
```

This expression states that the parameters $x_{1,1}, ...$ use the random walk as proposal strategy and the proposal distribution for them is Norm(0, 0.1).

4.2.5 The update strategy and the number of updates

The format of the file depends on the update strategy to be chosen.

1) If the user wants to update the parameters in sequential order:

```
? update
sequential

? iterations
42
```

The user gives the number of iterations, that is: the parameters are updated once and then the output is printed (considering the thinning factor of course) and this is repeated as many times as the iteration count.

2) If the user wants the next parameter to update to be chosen at random:

```
? update
random

? updates
```

```
500

? x 2:5 2:3
600
```

The user gives first gives the update strategy, then the default number of updates. Considering this example all parameters must be updated at least 500 times before the simulation is finished. After giving the default number, the user can also give the number of updates for single parameters. Note: The output is printed after each update so the user may want to use a bigger thinning factor.

4.2.6 Which parameters to output

The parameters to output are defined in a manner similar to the initial values. For example:

```
alpha all
beta 2:2
gamma
```

This definition states that the parameter to output are $alpha_1, ..., alpha_5, beta_2$ and $gamma$.

5 Generated program

5.1 Data structures

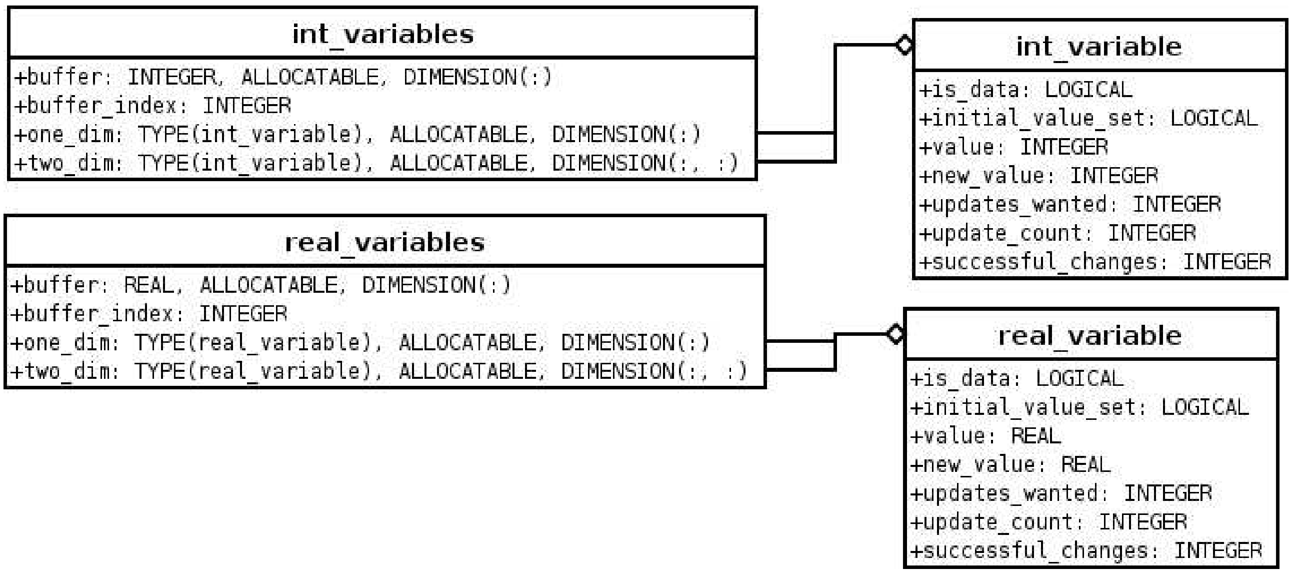
int_variables
+buffer: INTEGER, ALLOCATABLE, DIMENSION(:)
+buffer_index: INTEGER
+one_dim: TYPE(int_variable), ALLOCATABLE, DIMENSION(:)
+two_dim: TYPE(int_variable), ALLOCATABLE, DIMENSION(:, :)
real_variables
+buffer: REAL, ALLOCATABLE, DIMENSION(:)
+buffer_index: INTEGER
+one_dim: TYPE(real_variable), ALLOCATABLE, DIMENSION(:)
+two_dim: TYPE(real_variable), ALLOCATABLE, DIMENSION(:, :)
int_variable
+is_data: LOGICAL
+initial_value_set: LOGICAL
+value: INTEGER
+new_value: INTEGER
+updates_wanted: INTEGER
+update_count: INTEGER
+successful_changes: INTEGER
real_variable
+is_data: LOGICAL
+initial_value_set: LOGICAL
+value: REAL
+new_value: REAL
+updates_wanted: INTEGER
+update_count: INTEGER
+successful_changes: INTEGER

Figure 3: A diagram of the generated data structures.

Figure 3 shows the data structures used in the generated program. The *int_variable* and *real_variable* can represent both data variables and parameters. Each instance corresponds for example to one $alpha_{3}2$ or $x_{3}1_{4}$. A *variables* instance represents a repetitive structure of the model, for example $alpha$ and x. The *one_dim* and *two_dim* are used for one- and two-dimensional variable structures, respectively.

5.2 Modules

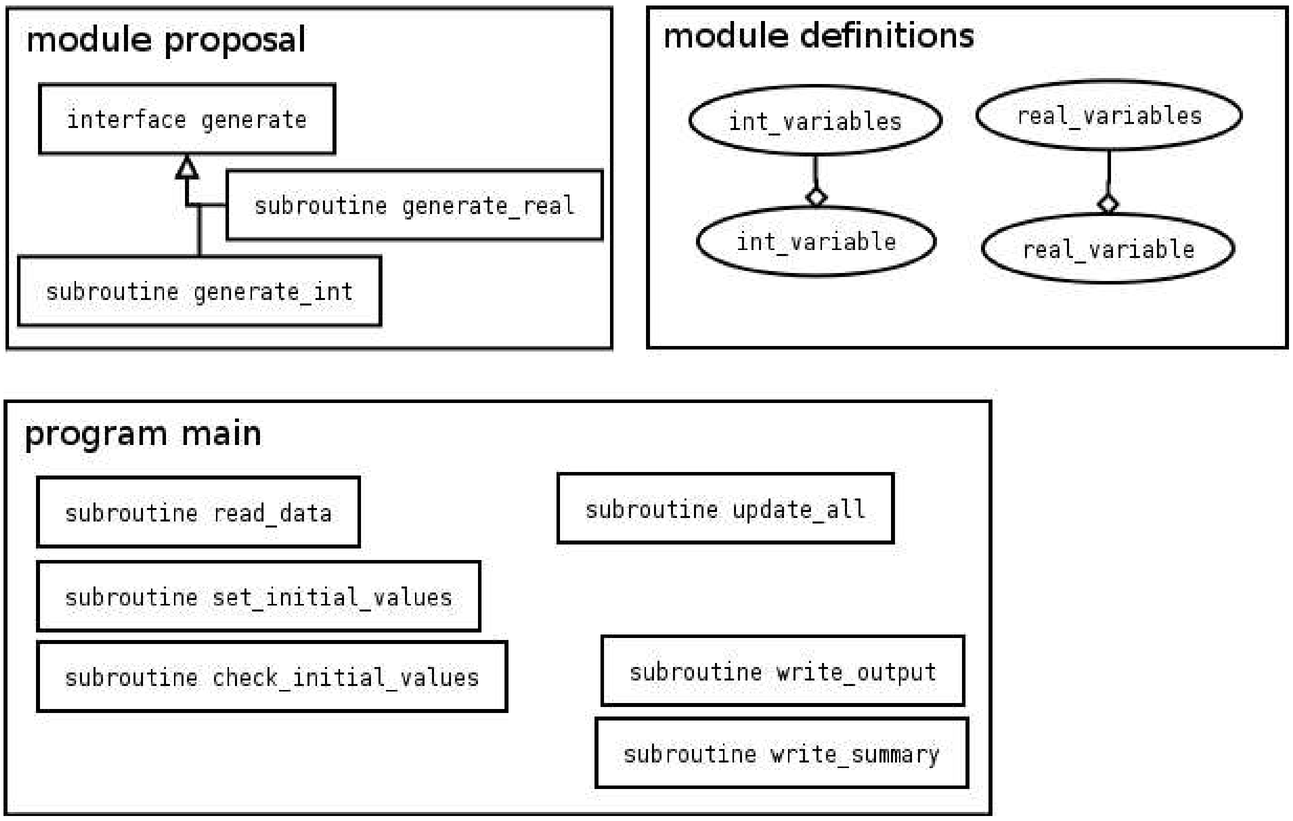

Figure 4: A diagram of the generated modules and their subroutines.

Figure 4 illustrates the division of the generated program into modules. Each module is placed in its own source file.

5.2.1 Module proposal

Subroutine name:	generate_int
Description:	Generates a buffer of new proposals for a given variable by its name.
Parameters:	CHARACTER(LEN=*), INTENT(IN) :: name INTEGER, DIMENSION(:), INTENT(OUT) :: buffer 1. name: *On entry:* The name of the variable to generate proposals for, e. g. $alpha$ 2. buffer: *On exit:* The buffer filled with new proposals from the variable's proposal distribution.
Generating:	All the names of the variable groups, their proposal distributions (names and parameters) and the corresponding functions/subroutines.

Subroutine name:	generate_real
Description:	
Parameters:	...
Generating:	See generate_int

5.2.2 Program main

Program name: main

Description:

Generating:

- The names and types (real or integer) of the variables in the model
- The name of the output file
- The loop lengths, that is, how many different entities (birds, squares etc.) we have
- The number of iterations, the thinning factor and the burn-in iteration count

Subroutine name: read_data

Description: Reads the data from data files into the data structure defined in figure 3.

Parameters: None. This subroutine uses the global variables defined in the beginning of the main.

Generating:

- The names of the data files
- The maximum length of a line in all the data files - this is needed because reading matrices that may contain ‘no data’-characters as well as numbers is difficult in Fortran
- The names of the variables and whether they are one-dimensional or two-dimensional
- The loop lengths, that is, how many different entities (birds, squares etc.) we have

Subroutine name:	set_initial_values
Description:	This subroutine reads the initial values from a data file into the data structure defined in figure 3.
Parameters:	None. This subroutine uses the global variables defined in the beginning of the main.
Generating:	The name of the initial values file. The names of the variable groups and whether they are one-dimensional or two-dimensional.

Subroutine name:	check_initial_values
Description:	This subroutine checks if the initial values of all parameters (not found in data) are given. If not, the simulation cannot start.
Parameters:	None. This subroutine uses the global variables defined in the beginning of the main.
Generating:	The names of the variable groups and whether they are one-dimensional or two-dimensional.

Subroutine name:	update_all
Description:	This subroutine updates each parameter once. It occurs in the generated program if and only if the user has chosen the sequential update strategy.
Parameters:	None. This subroutine uses the global variables defined in the beginning of the main.
Generating:	• The names of the parameters and whether they are one-dimensional or two-dimensional • The loop lengths, that is, how many different entities (birds, squares etc.) we have • The proposal strategy (fixed or random walk) for each parameter • The formula of the acceptance probability for each parameter (which variables depend on it and which parameter depend on it and what distributions define the dependencies • The proposal distributions for each parameter

Subroutine name:	write_output
Description:	This subroutine writes the output of one iteration into the output file. The file is opened and closed in the main program.
Parameters:	None. This subroutine uses the global variables defined in the beginning of the main.
Generating:	Which parameters to write as output and whether they are one- or two-dimensional.

Subroutine name:	write_summary
Description:	This subroutine writes the summary of the simulation into a summary output file. The summary includes the number of updates and successful changes for each parameter.
Parameters:	None. This subroutine uses the global variables defined in the beginning of the main.
Generating:	The names of the parameters and whether they are one- or two-dimensional. The file name of the summary file.

Subroutine name:	
Description:	
Parameters:	...
Generating:	

6 Component interfaces

This section will describe the ways of interaction between the generator, the executable program, and input files.

7 Modules

This chapter will divide the generator into logical Java classes and the executable program prototype into Fortran modules. There will be diagrams and descriptions.

8 References

Rand	*Definition of Random Fields in Encyclopedia* http://encyclopedia.laborlawtalk.com/Random_fields
Tode04	Pekka Tuominen: Todennäköisyyslaskenta I
Math05	http://mathworld.wolfram.com/