Distributed computing of failure probabilities for structures in civil engineering

Andrés Wellmann Jelic, University of Bochum (andres.wellmann@rub.de)
Matthias Baitsch, University of Bochum (matthias.baitsch@rub.de)
Dietrich Hartmann, University of Bochum (hartus@inf.bi.rub.de)
Karsten Spitzlei (spitzlei@inf.bi.rub.de)
Daniel Ballnus (ballnus@inf.bi.rub.de)

Abstract

In this contribution the software design and implementation of an analysis server for the computation of failure probabilities in structural engineering is presented. The structures considered are described in terms of an equivalent Finite Element model, the stochastic properties, like e.g. the scatter of the material behavior or the incoming load, are represented using suitable random variables. Within the software framework, a Client-Server-Architecture has been implemented, employing the middleware CORBA for the communication between the distributed modules. The analysis server offers the possibility to compute failure probabilities for stochastically defined structures. Therefore, several different approximation (FORM, SORM) and simulation methods (Monte Carlo Simulation and Importance Sampling) have been implemented. This paper closes in showing several examples computed on the analysis server.

1 Introduction

All structures in civil engineering contain various uncertain properties. Modern standards customarily represent uncertainties in terms of semi-probabilistic checking concepts. By contrast, a full probabilistic reliability analysis can be performed, which determines the failure probability of the system based on the stochastic properties for specified limit states. These limit states are described and analyzed independently from the scatter of the parameters x used in the correspondent limit state function $g : x \to g(x), x \in \Re^n$. Standard methods in a reliability analysis compute results of this function $g(x)$ in order to estimate the failure probability, independent on the type of problem definition. This independency permits a separate implementation of the reliability analysis. Therefore, the correspondent software module would be applicable for different fields of interest, like e.g. dynamical reliability problems or design in foundation engineering. In the framework of this contribution, the server module has been developed and adapted to solve reliability problems of structural systems.

The developed software has to fulfill several important requirements. The first main requirement is the modularity of the implementation in order to facilitate the exchange or addition of further methods and problem definitions. Emphasis is also placed on the reusability of this module to make it easily adaptable in subsequent analyses. As the execution of reliability analyses is very time consuming, a distributed software design is favorable. The calculations can be executed on high-performance servers and controlled via a client on any standard computer. The first two requirements can be satisfied using the object oriented programming language JAVA (SUN 2003). The distributed implementation is achieved by passing the communication between the separated modules in terms of the middleware CORBA. The technical implementation of CORBA is applying the *Object Request Broker (ORB)* from *Object Oriented Concepts, Inc.*(IONA-Techologies 2001).

It is possible to integrate several methods for the estimation of failure probabilities, leaving the choice of approach to the user. An overview of the implemented calculations methods is given in chapter 2. The software design and the implementation in JAVA of the analysis server and additional components will be explained in chapter 3. In chapter 4, the usage of the server modules is explained with several examples in the field of structural engineering. This contribution finishes with a short conclusion.

2 Probabilistic Analyses

The main goal of a probabilistic analysis is the estimation of the probability of failure for a structural system by evaluating the multidimensional function

$$P_f = \int\limits_{g(x)\leq 0} f(\mathbf{x})\mathbf{dx} \tag{1}$$

In Eq.(1) a joint probability density function of the vector $\mathbf{x}$ of random variables has to be integrated in the failure domain. This domain is defined by the so-called limit state function $g(x)$, indicating failure with negative values and survive with positive values. In some cases, the limit state function is not defined in terms of the random variables x explicitly. In particular, it may depend only on response quantities computed in a Finite Element analysis.

A straightforward numerical solution of the integral function is only feasible in some special cases. Instead simulation and approximation methods are used in order to estimate a solution.

2.1 Approximation methods

Well developed methods for approximating the failure probability are FORM and SORM (First-Order and Second-Order Reliability Methods). These are analytical solutions converting the integration into an optimization problem. In order to simplify the calculation the distribution functions of the random variables and the limit state function are transformed into a standardized Gaussian space. This transformation is defined via the *Cumulative Distribution Function*

$$F_{X_i}(x_i) = \Phi(y_i) \tag{2}$$

where y_i are the transformed and standardized Gaussian variables, leading to

$$y_i = \Phi^{-1}(F_{X_i}(y_i)). \tag{3}$$

This transformation leads to a nonlinear limit state function

$$h(y) = g\left(F_{X_1}^{-1}\left(\Phi(x_1)\right), \ldots, F_{X_m}^{-1}\left(\Phi(x_m)\right)\right) \tag{4}$$

in almost all cases. The FORM and SORM now simplify these functions calculating linear and quadratic tangential surfaces respectively. These surfaces are adapted in the so-called *design point y^**. This point of the limit state function $h(y)$ is defined via the shortest distance (e.g. in FORM)

$$\delta = \frac{h(\mathbf{y}) - \sum\limits_{j=1}^{m} y_j \frac{\partial h}{\partial y_j}}{\left(\sum\limits_{j=1}^{m}\left(\frac{\partial h}{\partial y_j}\right)^2\right)^{1/2}} \tag{5}$$

between $h(y)$ and the coordinate origin of the standardized Gaussian space. From this distance measure the safety index

$$\beta = \begin{cases} +\delta, & h(\mathbf{0}) > 0 \\ -\delta, & h(\mathbf{0}) < 0 \end{cases} \tag{6}$$

is derived. This leads to a simplified formulation of the failure probability

$$P_f \approx \Phi(-\beta) \tag{7}$$

in FORM and to

$$P_f = \Phi(-\beta) \prod_{i=1}^{m-1} (1 - \beta\kappa_i)^{-1/2} \tag{8}$$

in SORM. The most time-consuming part in these methods is finding the design point. Several iterations have to be calculated until the distance measure δ shows good convergence.

2.2 Simulation methods

In contrast to the approximation methods named above the class of *Monte Carlo Simulations* has to be mentioned. These methods use the given density functions to create multiple sets of realizations of all random variables. For each set of realizations, a deterministic analysis of the researched limit state function $g(x)$ is performed, in our case, a structural analysis using the Finite Element Method. Afterwards, the results are evaluated concerning failure or survival. In order to simplify the description of the analysis results an *Indicator function*

$$I(g(\mathbf{x})) = \begin{cases} 1, & \text{für } g(\mathbf{x}) < 0 \\ 0, & \text{für } g(\mathbf{x}) \geq 0, \end{cases} \tag{9}$$

is used. This leads to an alternative formulation of the failure probability in Eq.(1)

$$P_f = \int_{-\infty}^{\infty} I(g(\mathbf{x})) \cdot f_X(\mathbf{x}) \mathbf{dx}. \tag{10}$$

In a discrete simulation this can be reduced to the finite sum

$$P_f = \frac{1}{n} \sum_{i=1}^{n} I[g(\mathbf{x}_i) < 0] \tag{11}$$

with n is describing the number of simulations and $\mathbf{x}_i$ is the i-th set of generated realizations. The big disadvantage of the classical Monte Carlo Simulation is that the accuracy of the estimated results are proportional to $1/\sqrt{n}$. Therefore , an increase of accuracy by one order of magnitude demands an increased execution of discrete simulations by around two orders of magnitude. The main reason is the clustered generation of realizations of the random variables near their mean values. As the demanded failure probabilities in structural engineering are very small, an uneconomic number of simulations have to be performed intending to get good estimations.

Consequently, the class of variance reducing methods have been developed based on the classic Monte Carlo Simulations. Some variations are e.g *Importance Sampling, Stratified Sampling or Adaptive Sampling*, more details can be found in (Bucher 1988; Schuëller 1998). So far, only *Importance Sampling* has been implemented in the given software package. Therefore, the main principles will be explained shortly. The *Importance Sampling* method moves the main generation point for realizations near the *design point* y^*, shown in Eq.(5), and then defines a new simulation density function $h(\mathbf{v})$ in y^*. This expands the integral in Eq.(10) to

$$P_f = \int \cdots \int I(\mathbf{v}) \frac{f_{\mathbf{x}}(\mathbf{v})}{h_V(\mathbf{v})} h_V(\mathbf{v}) \, \mathbf{dv}. \tag{12}$$

Hence, the failure probability can be estimated with

$$P_f = \frac{1}{m} \sum_{n=1}^{m} I(\mathbf{v}_n) \frac{f_{\mathbf{x}}(v_n)}{h_V(\mathbf{v}_n)} \tag{13}$$

using m simulation runs and the sample $\mathbf{v}_n$ defined by h. In order to calculate approximate estimates for the failure probability a good choice of the sampling density $h(\mathbf{v})$ is essential. The variance of Eq.(13) is

$$\text{Var}[P_f] = \frac{1}{m-1} \left[\frac{1}{m} \sum_{n=1}^{m} I(\mathbf{v}_n) \left(\frac{f_{\mathbf{x}}(v_n)}{h_V(\mathbf{v}_n)} \right)^2 - P_f^2 \right], \tag{14}$$

leading to a coefficient of variance

$$v_{P_f} = \frac{(\text{Var}[P_f])^{1/2}}{P_f}. \tag{15}$$

The exact solution for P_f is obtained for a proportional definition of $h_V(\mathbf{v})$ to the real density function $f_X(\mathbf{v})$, which, however, implies the knowledge of the searched probability. Instead, (Spaethe 1992) proposes, therefore, the use of the original density function of $f_V(\mathbf{v})$, a normal or a uniform distribution. In the developed software only the use of the normal or uniform distribution is offered. Subsequently, the *design point* is determined from a pre-executed FORM calculation.

3 Software design

For the estimation of failure probabilities of a system the approach can be subdivided into the problem definition and the reliability analysis.

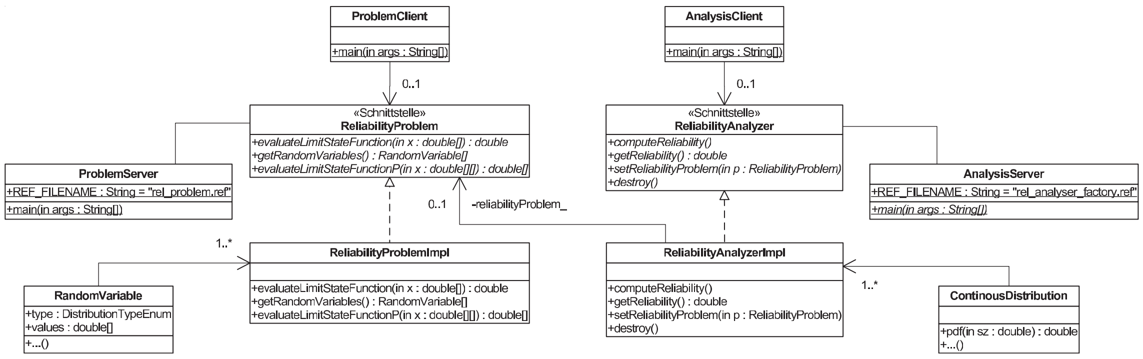

ProblemClient
+main(in args : String[])
0..1
«Schnittstelle»
ReliabilityProblem
+evaluateLimitStateFunction(in x : double[]) : double
+getRandomVariables() : RandomVariable[]
+evaluateLimitStateFunctionP(in x : double[][]) : double[]
ProblemServer
+REF_FILENAME : String = "rel_problem.ref"
+main(in args : String[])
0..1
-reliabilityProblem_
1..*
ReliabilityProblemImpl
+evaluateLimitStateFunction(in x : double[]) : double
+getRandomVariables() : RandomVariable[]
+evaluateLimitStateFunctionP(in x : double[][]) : double[]
RandomVariable
+type : DistributionTypeEnum
+values : double[]
+...()
AnalysisClient
+main(in args : String[])
0..1
«Schnittstelle»
ReliabilityAnalyzer
+computeReliability()
+getReliability() : double
+setReliabilityProblem(in p : ReliabilityProblem)
+destroy()
AnalysisServer
+REF_FILENAME : String = "rel_analyser_factory.ref"
+main(in args : String[])
ReliabilityAnalyzerImpl
+computeReliability()
+getReliability() : double
+setReliabilityProblem(in p : ReliabilityProblem)
+destroy()
1..*
ContinousDistribution
+pdf(in sz : double) : double
+...()

Figure 1: Overview of software design

This differentiation is represented in the software design by the separate implementation of problem and analysis (see Fig.1). Main controlling structures in this software are the two server modules *ProblemServer* and *AnalysisServer*. These objects create and manage the correspondent parts of the reliability analysis, a *ReliabilityProblem* and the *ReliabilityAnalyzer*, which are defined as interfaces in the IDL language for CORBA definitions and implemented in JAVA. The clients access these objects via the IDL-defined interfaces.

3.1 Server module for problem definition

The first part of the approach includes all stochastic properties of the system. Furthermore, the considered system itself has to be included in a proper mathematical way. In civil engineering e.g. the scatter of material properties and the stochastic behavior of loads can be collected and described in a set of structural random variables $\mathbf{X}$. The structural system could be represented using the *Finite Element Method*. These parameters form the input of the *limit state function* (Eq.(4)) which describes the failure mechanism of the system in a mathematical way. These relations are displayed in Fig.2.

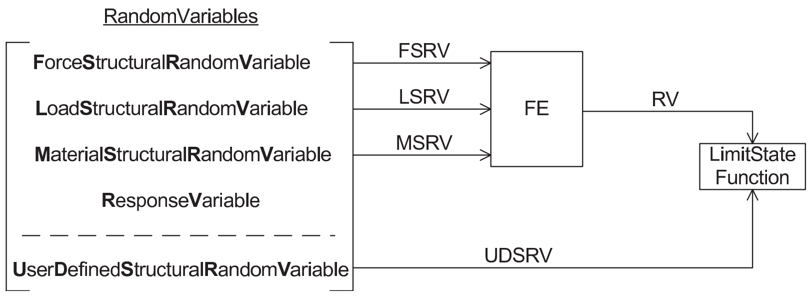
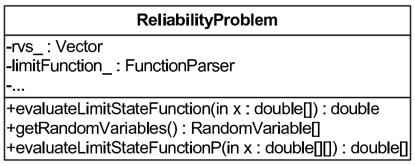

RandomVariables
ForceStructuralRandomVariable
LoadStructuralRandomVariable
MaterialStructuralRandomVariable
ResponseVariable
UserDefinedStructuralRandomVariable
FSRV
LSRV
MSRV
FE
RV
LimitState Function
UDSRV

Figure 2: Parameters of a structural reliability problem

The parameters and functions mentioned above are necessary in order to define a *Reliability Problem* which is created by the server module for problem definition. This definition can be used later in the subsequent reliability analysis. Fig.3 shows a general reliability problem.

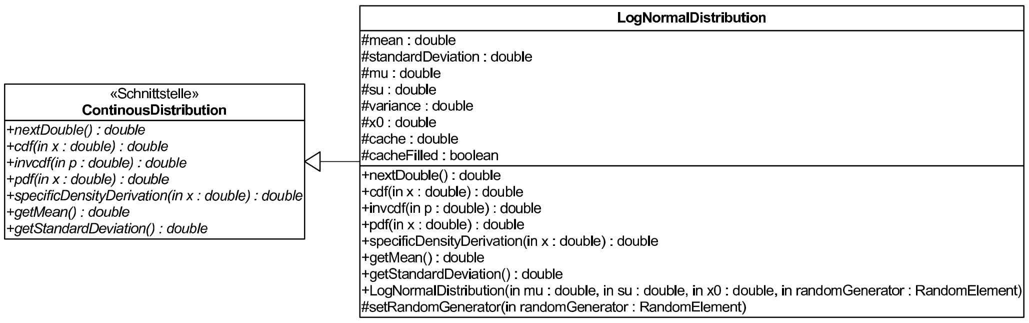

ReliabilityProblem
-rvs_ : Vector
-limitFunction_ : FunctionParser
-...
+evaluateLimitStateFunction(in x : double[]) : double
+getRandomVariables() : RandomVariable[]
+evaluateLimitStateFunctionP(in x : double[][]) : double[]
Figure 3: Structure of a general reliability problem

As already alluded previously, the problem definition has been adapted in this work to describe structural reliability problems in civil engineering. But, the communication of the server modules via the general module *Reliability Problem* allows the definition of different problems where failure probabilities have to be computed. The problem definition would only differ at the information of random variables and the limit state function $g(\mathbf{x})$.

3.2 Server module for analyses

The second part contains the necessary information in order to control and start the mathematical analysis of the reliability problem. This can be stored in a module named *Reliability Analyzer*. This module gets the stochastic input for the analysis via the above described module *Reliability Problem*. It also includes the calculation methods like e.g. FORM or *Importance Sampling*.

Furthermore, the module has to define one stochastic distribution function for each random variable in the the *Reliability Problem*. This functions will be used in the execution of the computation methods. The actual software version only offers the possibility to create the following types of continuous distributions: Normal, Lognormal, Exponential, Uniform, Gumbel, Frêchet and Weibull. Because of the demanded modularity of the software it will be possible to add further types of distribution functions without the necessity of changing the *Reliability Analyzer* structure itself. A typical design of such a distribution function, derived from a general interface *ContinousDistribution*, is shown in Fig.4.


«Schnittstelle»
ContinousDistribution
+nextDouble() : double
+cdf(in x : double) : double
+invcdf(in p : double) : double
+pdf(in x : double) : double
+specificDensityDerivation(in x : double) : double
+getMean() : double
+getStandardDeviation() : double
LogNormalDistribution
#mean : double
#standardDeviation : double
#mu : double
#su : double
#variance : double
#x0 : double
#cache : double
#cacheFilled : boolean
+nextDouble() : double
+cdf(in x : double) : double
+invcdf(in p : double) : double
+pdf(in x : double) : double
+specificDensityDerivation(in x : double) : double
+getMean() : double
+getStandardDeviation() : double
+LogNormalDistribution(in mu : double, in su : double, in x0 : double, in randomGenerator : RandomElement)
#setRandomGenerator(in randomGenerator : RandomElement)
Figure 4: Design of a continuous distribution

Finally, the *Reliability Analyzer* stores the results of the analysis in one real-valued number. The above explained relations are displayed in Fig. 5.

The described separation between analysis and problem definition allows to use the mathematical methods of reliability analysis without knowing what is represented by the problem definition. The server module for the analysis just needs access to the correspondent *limit state function* and the stochastic properties of each random variable (e.g. type of distribution $f(x)$, mean value m_x, standard deviation σ_x).

ReliabilityAnalyzer
-reliabilityProblem_ : ReliabilityProblem -failureProbability_ : double -method_ : MethodInternal -continousDistributionVector_ : ContinousDistributionVector -...

+computeReliability()
+destroy()
+getFailureProbability() : double
+setReliabilityProblem(in p : ReliabilityProblem)
+getReliabilityProblem() : ReliabilityProblem
+setMethod(in m : MethodTypeEnum)
+getMethod() : Method
+toString() : String
+getMethodInfo() : String
+setMethodCancelled(in b : boolean)
+isResult() : boolean
+ReliabilityAnalyzer(in orb : ORB)
+getContinousDistributionVector() : ContinousDistributionVector
+setFailureProbability(in f : double)

Figure 5: Definition of a reliability analyzer

3.3 Client modules

In order to control the computations each server module needs a correspondent client. In the first instance, the client for the problem definition has to be started. The main task is to collect all necessary information for the definition and to save this in a *Reliability Problem* object (see Fig.6). Furthermore, a client connected to the analysis server module has to be implemented. At initialization, this client creates an instance of a *Reliability Analyzer* object and connects this to a

ProblemClient
-reliabilityProblem_ : ReliabilityProblem -...

+ProblemClient(in reliabilityProblem : ReliabilityProblem)
+run()

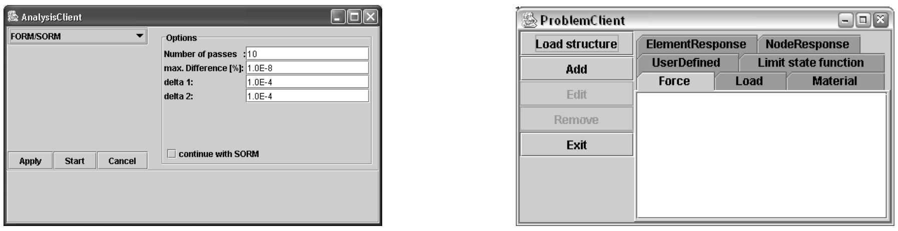
Figure 6: Problem client

AnalysisClient
-reliabilityAnalyzer_ : ReliabilityAnalyzer -method_ : Method -...

+AnalysisClient(in reliabilityAnalyzer : ReliabilityAnalyzer)
+initAction()
+printData()

Figure 7: Analysis client

given reliability problem (see Fig.7). Subsequently, it manages the selection and execution of calculation methods by the user via a graphical interface. When the calculation stops, it should show the correspondent results in a separate output area. The graphical interfaces for the clients, implemented in JAVA, are shown in Fig.8.


AnalysisClient

FORM/SORM

Options

Number of passes : 10
max. Difference [%]: 1.0E-8
delta 1: 1.0E-4
delta 2: 1.0E-4

☐ continue with SORM

ProblemClient

Figure 8: GUI for client modules

4 Examples

In this section three different examples of a reliability analysis in structural engineering are demonstrated. The results have been computed using the above presented software framework. Within each example the results from all implemented analysis methods will be compared. Also, the quality of the estimated results will be shown within the first example.

4.1 Example 1: Cantilever beam

This example is taken from (Spaethe 1992, p.78). A cantilever beam with a length of 2,0m is loaded with a singe vertical force P at the overhanging end (see Fig.9).

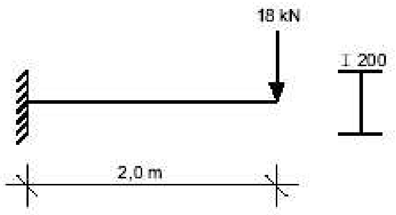


Figure 9: Cantilever beam

Height (cm)	I_y (cm^2)	A (cm^4)	W_y (cm^3)	E $(^{kN}/_{cm^2})$
20	2140	33,4	214	26500

Figure 10: Parameters of the cantilever beam

Fig.10 lists the data of the cross section and the material. The limit state function $g(x)$ is computed as the *Ultimate Limit State*, defined by the initiation of yielding in the outer fiber. The probability of failure is to be estimated for this limit state. The stress analysis is defined via

$$g(x) = \sigma_F \cdot 214 - \mid M \mid \leq 0. \tag{16}$$

Stochastically defined values are the yield point of the steel material (random variable X_1) and the single load P (random variable X_2). The stochastic informations are given in Fig.11.

i	m_{x_i}	σ_{x_i}	x_{0i}	Type of distribution
1	$26,5^{kN}/_{cm^2}$	$2,5^{kN}/_{cm^2}$	$16^{kN}/_{cm^2}$	log. Normal
2	$-18kN$	$2kN$	-	Gumbel, Smallest values

Figure 11: Statistical parameters

Based on these definitions all implemented calculation methods have been used to estimate the failure probabilities. The correspondent failure probabilities P_f and the absolute runtime of each method are summarized in Fig.12. The exact solution of this reliability problem, computed in (Spaethe 1992) via numerical integration of the multidimensional integral in Eq.(1), is equal to $2,131 \cdot 10^{-3}$.

Method	P_f	Duration
		[ms]^1
MCS (100.000 samples)	0,00246	500.700
MCS (5.000.000 samples)	0,00209	21.175.509
FORM	0,00224	521
SORM	0,00214	2.654
IS (normal distr.; 10.000)	0,00206	78.122
IS (equally distr.; 10.000)	0,00172	70.181

Figure 12: Calculation results

Comparing the results obtained from the *Monte Carlo Simulation* to the exact result, a good agreement can be stated only for a sample size of $n = 5.000.000$. A simulation run with only 100.000 samples, however, shows a variation of 10%. The quality of the estimations can be measured with the variance of the estimator, shown in Fig.13. At the left side, the change of the variance for 100.000 simulations depending on the number of passes is shown. Each pass consisted of 1.000 discrete samples. The figure on the right displays the variance for 5.000.000 simulations, each pass containing 10.000 discrete samples. A better convergence of the values is evidently achieved with an increasing number of simulations.

FORM and *SORM* lead to a good agreement with respect to the real value. The results from *FORM* having a variation of 5% are improved by means of a subsequent *SORM* run, to a

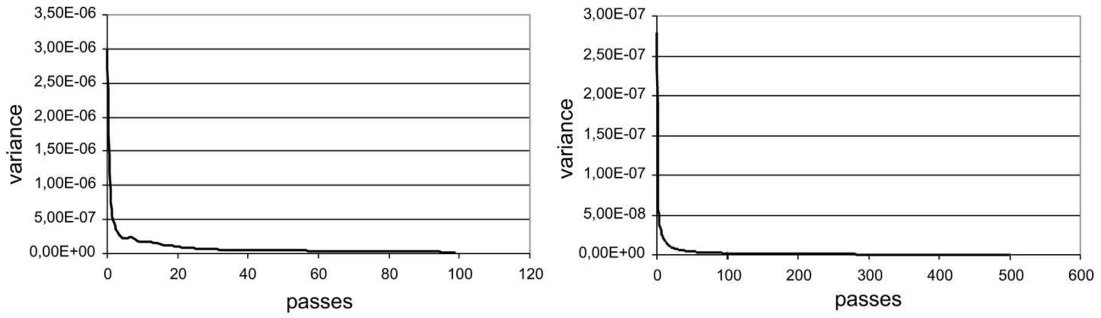


Figure 13: Decrease of the variance for 100.000 and $5.000.000$ samples

variation of only 1%. Finally, the *Importance Sampling* method delivers appropriate results with a variation of 3% (using a normal distribution), most importantly, within an acceptable runtime (see Fig.12). This method was performed twice using a equally distributed function and a normal distribution function in the design point. In Fig.14 the quality of the results is shown.

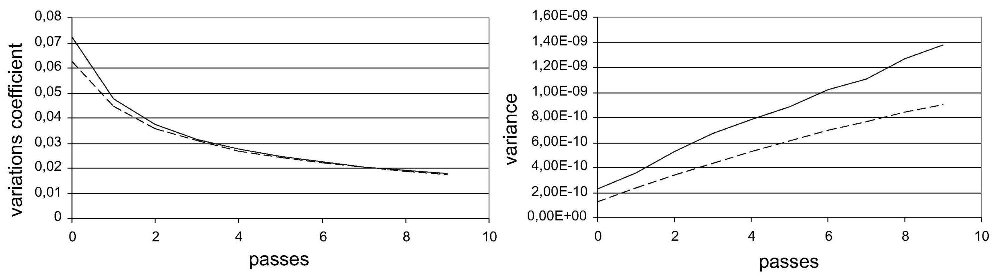


Figure 14: Coefficient of variation and variance for Importance Sampling using a normal distribution (—) and an uniform distribution (- - -)

The design point was searched with a FORM analysis. Afterwards, the *Importance Sampling* method is executed with 10.000 samples. This results in a decreasing coefficient of variation using a normal and an uniform distribution. Though the variance increases, it is still smaller than the variance in a *Monte Carlo Simulation* after 100.000 runs. Both calculations in Fig.14 show a good convergence, although the estimation of the failure probability is more exact using the normal distribution for the sampling density.

4.2 Example 2: Framed steel structure

A two-hinged frame from steel constructions is analyzed with respect to the *Ultimate Limit State*, proving the maximum yield stresses in the upper left haunch (see Fig.15). The values of the cross sections are given in Fig.16.

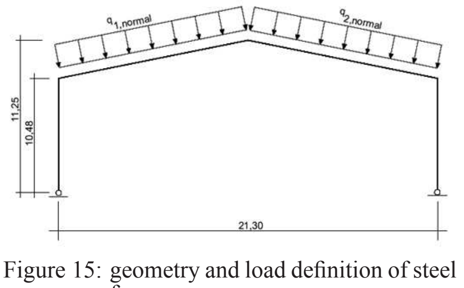


Figure 15: geometry and load definition of steel frame

El.	h	I_y	A	W_y	E
-	(cm)	(cm^2)	(cm^4)	(cm^3)	$({}^{kN}/{}_{cm^2})$
Col.	40	23128	$84,5$	$1156,4$	21.000
Bar	33	11767	$62,6$	$713,2$	21.000

Figure 16: Parameters of the framed structure

Again, the stochastically defined parameters are the yield point of the material (X_1) and the uniform load on the bar (X_2), both defined with a Log-Normal distribution. The load is the combination of the self weight of the structure and additional snow load $q_s = 0,0375\ ^{kN}/_{cm}$. The parameters of the distribution are summarized in the Fig.17.

i	m_{x_i}	σ_{x_i}	x_{0i}	Type of distribution
1	$24^{kN}/_{cm^2}$	$2,4^{kN}/_{cm^2}$	$14^{kN}/_{cm^2}$	log. Normal
2	$0,07001^{kN}/_{cm}$	$0,00375^{kN}/_{cm}$	$0,03251^{kN}/_{cm}$	log. Normal

Figure 17: Statistical parameters

The results of this example are only listed shortly in Fig.18. Comparing these results with the demanded reliability in the standard codes, defined in a range between 10^{-3} for SLS and 10^{-6} for ULS, it is obvious that the reliability of the given structure is too small. One possible improvement could be the usage of a cross section with higher bearing capacity, like e.g. an IPE 450 from the German code. The change of the cross section and subsequently also the self weight of the structure leads to the results displayed in Fig.19.

Method	Failure probability P_f
MCS	$1,101 \cdot 10^{-2}$
FORM	$1,09 \cdot 10^{-2}$
SORM	$0,997 \cdot 10^{-2}$
IS	$1,108 \cdot 10^{-2}$

Figure 18: Results from reliability analysis

Method	Failure probability P_f
MCS	$6,2 \cdot 10^{-6}$
FORM	$4,666 \cdot 10^{-6}$
SORM	$5,544 \cdot 10^{-6}$
IS	$5,612 \cdot 10^{-6}$

Figure 19: Results for improved structure

4.3 Example 3: Multi-story frame

In the last example, a different limit state is analyzed. The structure is a multi-story frame from steel constructions (see Fig.20) with several horizontal loads.

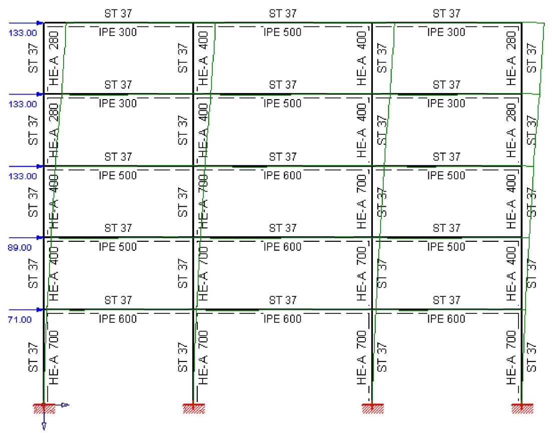


Figure 20: Framed structure and cross sections

The *limit state function*

$$g(x) = \frac{h}{320} - u_{24} \leq 0 \tag{17}$$

describes the maximum horizontal displacement of the upper right corner. Stochastically defined entities are the load P_i and the modulus of elasticity E (E_1 for the horizontal, E_2 for the vertical trusses), shown in Fig.21.

A reliability analysis of this structure returns the result listed in Fig.22.

Variable	distribution type	Unit	m_x	σ_x	a	u
P_1	Gumbel, small	kN	$-220,0$	$80,0$	$0,016032$	$-183,99673$
P_2	Gumbel, small	kN	$-170,0$	$75,0$	$0,0171007$	$-136,2469$
P_3	Gumbel, small	kN	$-133,0$	$70,0$	$0,0183221$	$-101,49714$
E_1	Normal	m^4	$2,17375E7$	$1,9152E6$	-	-
E_2	Normal	m^4	$2,37963E7$	$1,9152E6$	-	-

Figure 21: Statistical parameters of example 3

Method	Failure probability P_f
MCS	$1,0 \cdot 10^{-6}$
FORM	$3,27517 \cdot 10^{-6}$
SORM	$1,05869 \cdot 10^{-6}$
IS	$2,7804 \cdot 10^{-6}$

Figure 22: Results for example 3

5 Conclusions

In this paper, a distributed software framework for carrying out reliability analyses is proposed. The main module within this framework is the implementation of a *Reliability Analyzer*, a tool for the initiation and management of a reliability analysis. In order to compute the failure probability, the indicator value for the reliability, several approximation and simulation methods have been presented and implemented. Furthermore, continuous distribution functions for the representation of stochastic properties have been included.

The analysis module has been used in association with a server module for the problem definition where the information about the reliability problem is stored. Furthermore, the server module for problem definition has been adapted in this work to solve structural reliability problems. It could be demonstrated that the software framework is useful to solve different representative examples in which the computations of time-independent failure probabilities is essential.

The main advantage of the presented concept is the strict separation between the problem definition and its analysis. By that, it is possible to apply the *Reliability Analyzer* in different reliability scenarios. Additionally, the modularity allows the supplementary inclusion of improved computational methods and distribution functions, if desired or available.

6 Endnotes

1) All calculations were performed on a PC System: Win XP, AMD Duron 800MHz

7 References

Bucher, C. (1988). Adaptive sampling - an iterative fast monte carlo procedure. *Structural Safety 5*(3), 119–126.

IONA-Techologies (2001). *ORBacus for C++ and JAVA. v4.1.0 API Specification.* http://www.iona.com/products/orbacus_home.htm.

Schuëller, G. (1998). Structural reliability - recent advances - freudenthal lecture. *Proceedings of the 7th International Conference on Structural Safety and Reliability (ICOSSAR'97) 1*, 3–35. A.A. Balkema Publications, Rotterdam, The Netherlands.

Spaethe, G. (1992). *Die Sicherheit tragender Baukonstruktionen*. Springer, Vienna.

SUN (2003). *JAVATM 2 Platform, Standard Edition, V.1.4.1 API Specification*. http://java.sun.com/j2se/1.4.1/.

Acknowledgements
This research is supported by the German Research Foundation (DFG) within the scope of the Collaborative Research Center 398 which is gratefully acknowledged by the authors.