Performance Characterisation and Verification of JavaSpaces based on Design of Experiments

Frederic Hancke, Tom Dhaene and Jan Broeckhove
Department of Mathematics and Computer Science
Middelheimlaan 1, 2020 Antwerp, Belgium
frederic.hancke@ua.ac.be

Abstract

In the ever increasing world of distributed systems, different middleware implementations can be compared qualitatively or quantitatively. Existing evaluation techniques are often not satisfying. In this contribution we apply Design of Experiments (DoE) to evaluate and model the performance of JavaSpaces, a Tuple Spaces implementation for distributing tasks through a virtual space. DoE is a statistical technique for identifying relevant inputfactors from a (large) set of inputfactors. The setup of the experiments is determined using Experimental Design (ED) and a selection of the relevant inputfactors is based on Analysis of Variance (ANOVA). Then Regression Analysis (RA) is used to obtain a multivariate representation. Some extra experiments are performed to validate this approach.

1. Introduction

Currently, many distributed platforms provide the resources required to execute a scientific application efficiently. The choice of which specific platform to use is up to the application programmer and is mostly based on the availability (open source), user-friendliness, etc. Often, these applications require high performance. This means that decent performance evaluation techniques are needed.

On the one hand, much research has been done in the field of performance *prediction* in parallel and distributed computing [17, 2]. On the other hand, only few research results on performance *evaluation* can be found in literature. Prediction is interesting when trying to optimize the distribution of applications, while evaluation is important particularly when deciding which distributed platform to use. Evaluation and comparison of distributed platforms are often realised through execution of some computationally expensive, complex but well-known applications, such as the calculation of Pi [1], generation of prime numbers [14], etc. Results of these experiments on different platforms are then compared using basic statistical techniques.

The goal of this contribution is to apply a predetermined set of statistical techniques to model the performance of a distributed platform. This set of techniques is often called *Design of Experiments* (DoE) [6, 10, 16]. The basic idea of DoE is to obtain the best possible characterisation while executing as few experiments as possible. Until now, this technique has been widely and successfully used in the engineering and chemistry world [15].

This contribution will first provide some background information on the Tuple Spaces paradigm [8] and an overview of the JavaSpaces implementation [7], followed by a description of DoE and its set of statistical techniques. Then, the setup and input- and outputfactors for DoE are discussed, followed by the presentation of the results of the application of DoE on the results of the experiments, and finally the resulting models are verified with additional experiments.

2. Tuple Spaces

The concept of *Tuple Spaces* was introduced in 1982 by the language *Linda* [8]. Linda is commonly described as a coordination language, developed to manage and control the coordination aspects of an application or algorithm (usually parallel). Linda does not constitute a full programming language in itself, but is designed to expand the semantics of an existing programming language in order to form a *parallel programming language*. Examples of this are the C-Linda [12], Fortran-Linda and Prolog-Linda [20] languages, respectively augmentations of the C, Fortran and Prolog programming languages.

The Linda language provides only a set of simple operations needed to operate on the contents of a virtual shared memory, the Tuple Space. The data structures contained in this shared memory are called *tuples*, which can be manipulated by the aforementioned set of operations. The three most important operations for the rest of this paper are:

- **out(t)** which writes a tuple *t* into the Tuple Space,
- **in(t)** which returns a tuple from the Tuple Space that matches a given template *t* (and thereby deletes *t* from the Tuple Space), and
- **rd(t)** which performs essentially the same actions as **in**, but keeps *t* in the Tuple Space.

These operations call for the definition of the format of the datastructures of the Tuple Space, the tuples themselves. Basically, a tuple is an ordered sequence of *fields*, each having a well determined type. Additionally, a Tuple Space does not require all tuples to have the same composition. This allows for the definition of *template tuples*: meta-tuples which describe the different possible type-based composition of normal tuples.

Linda implementations must guarantee the atomicity of the previously mentioned operations. Sun Microsystems' *JavaSpaces* specification [19] builds upon Jini [5] and RMI [18]. The goal of JavaSpaces [7] is to allow the creation of robust distributed systems in general, and distributed algorithms in particular. These systems are modelled as a set of distributed components communicating by passing messages through a space. A message or *entry* is represented by a Java object that can be stored in the space (i.e. a distributed shared memory). JavaSpaces provides the three basic operations *read*, *write* and *take*.

3. Design of Experiments

The Design of Experiments (DoE) technique combines the power of the advantages of multiple statistical techniques, i.e. *Experimental Design* (ED), *Analysis of Variance* (ANOVA) and *Regression Analysis* (RA) (see Fig. 1). *Experimental Design* [10, 11] is a technique for planning experiments for a system under test. After the execution of this experiment it is possible to analyse the results using *Analysis of Variance* and *Regression Analysis* [13]. These techniques allow for a *Response Surface Model* (RSM) [21] to be built.

3.1. Experimental Design

Experimental Design (ED) defines the experiment that must be executed to model a system. A *process* is a combination of machines, methods, physical reactions, human beings, etc. that, given an input and a set of inputfactors, generates an output where the values of a set of all outputfactors (also called responses) can be derived. This system acts as a *black box*. Among the inputfactors one can distinguish controllable and uncontrollable inputfactors. Controllable inputfactors are changeable at any time, independent of other factors, such as the number of machines participating in a distributed environment. Uncontrollable inputfactors are not independently changeable, such as the weather.

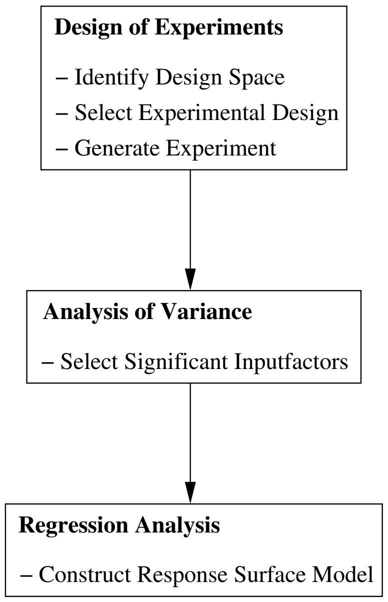

Figure 1. Overview of Design of Experiments (DoE)

An *experiment* is referred to as a set of tests to be executed. The levels of the inputfactors are changed appropriately during the ED phase such that changes in values of outputfactors can be observed and identified. Therefore, these factors must be measurable. For each of these tests, each controllable inputfactor is assigned a level, or numerical value in our case. The goal of ED is to set up as few tests as possible in an experiment. The choice of these tests is based on replication, randomisation and fixed levels for inputfactors. Replication of every test in an experiment leads to an experimental error. This error is caused by the unknown uncontrollable inputfactors. The order of execution of all tests with its replicates are randomized. This way, influences of unknown uncontrollable factors on the outputfactors can be avoided.

Every experiment is generated using a *model*. This model must be consistent with the target of the experiment and therefore must not be chosen too simple (e.g. linear when the output is expected to be quadratic). On the other hand, a model chosen too complex leads to a greater amount of tests to execute. A linear model is always a good model to start with. Using this, linear effects and interaction effects of the inputfactors on the outputfactors may be detected and analysed. If the linear model is not satisfying, more tests can be defined for a more complex model, such as a quadratic model.

The *Central Composite Circumscribed* (CCC) design [10] is a quadratic model extending the linear 2^n *Full Factorial* design [10] (including the central point) with tests on the starpoints. In a full factorial design all inputfactors are assigned two levels: *high* (or $+1$) and *low* (or -1). Optionally the central point can be tested, which is called the *central* level (or 0). The CCC design extends this linear model with two levels: $-\delta$ and $+\delta$ ($\delta = (2^n)^{1/4}$). These levels lie beyond the levels -1 and $+1$. An experiment generated using this CCC design contains $m(2^n + 2n) + m_c$ number of tests (with m the number of replicates, n the number of controllable inputfactors, and m_c the number of replicates of the central point). Figure 2 illustrates the CCC design for $n = 2$.

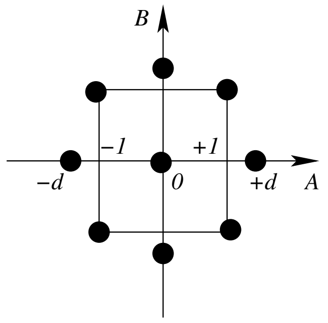

Figure 2. Representation of the testpoints of a Central Composite Circumscribed design with inputfactors A and B

Using the CCC design, a model can be built. For $n = 2$, x_A the value of inputfactor A, and x_B the value of inputfactor B, this model is given by

$$y = \gamma_0 + \gamma_1 x_A + \gamma_2 x_B + \gamma_{11} x_A^2 + \gamma_{22} x_B^2 + \gamma_{12} x_A x_B + \epsilon \quad (1)$$

This allows to predict outputfactor Y with respect to inputfactors A and B. The regression analysis will find the best possible fit to this model.

3.2. Analysis of Variance and Regression Analysis

After the designed experiment has been executed, the results (the values of the input- and outputfactors) can be fitted into the model allowing to obtain the model parameters. To fit this model, the *Analysis of Variance* (ANOVA) with a *Regression Analysis* can be used [13].

ANOVA will test hypotheses for each (combination of) inputfactor(s). H_0 supposing the factor to be 0, H_1 supposing the factor not to be 0. If H_0 holds, the inputfactor A is considered to have no influence on outputfactor Y. If not (meaning the alternative H_1 holds), A is considered to influence Y at some testpoint. To test whether H_0 holds, a *Level of Significance* (LoS) must be chosen. The LoS indiciates the risk of concluding H_0 is false, while being true in reality. Usually the LoS is taken five percent. ANOVA is able to select the most significant inputfactors (for which H_0 is rejected). Using these factors a *Response Surface Model* (RSM) can be built by performing a regression analysis with the least squares method. This regression will result in approximations of the γ_i parameters in (1).

4. Processmodel Parameters

In order to model the performance of JavaSpaces, a number of parameters for the process model must be defined that can then be used as inputfactors for the Experimental Design phase. This process model for JavaSpaces is based on the *Master-Slave* pattern. This pattern is widely used in distributed systems and is based on the idea of 1 master controlling a slaves. The master distributes b tasks to its slaves and collects their returned results. A task is defined as a problem that takes some time to solve. Suppose that all tasks need the same solving time c, then the total theoretical time T_t needed to solve all tasks is given by:

$$T_t = bc$$

In order to obtain full control of this process the concept of *Tunable Abstract Problem Profiles* (TAPPs) was developed [9]. TAPPs are based on dummy *sleep* tasks. These sleep tasks put the slaves' processor to sleep for a specific amount of time instead of calculating a real problem. This allows for much better prediction of the total theoretical time T_t of an experiment and the prediction of the overhead of a distributed platform. The wallclock time is defined as the time needed by the master between distributing the first task to one of its slaves and receiving the last result from one of its slaves. For JavaSpaces this wallclock time is denoted by T_{JS}.

The definition of the inputfactors for the process model is now straightforward: A indicates the number of slaves a, B the number of tasks b, and C the time c to *solve* one task[1].

Besides T_{JS}, another outputfactor, the *speedup* (S_{JS}), will complete the process model:

$$S_{JS} = \frac{T_t}{T_{JS}} \quad (2)$$

1 Remark all tasks are supposed to be independent.

5. The Experiment

5.1. Setup

Inputfactor	$-\delta$	-1	0	$+1$	$+\delta$
A	3	5	8	11	13
B	56	296	648	1000	1240
C	56	296	648	1000	1240

Table 1. Level values for each inputfactor (A: number of slaves; B: number of tasks; C: execution time of one task in ms)

As discussed in the previous section the experiment contains three inputfactors. Table 1 summarizes the level values for each of these factors[2]. The master process ran on an *Intel PIII 733MHz* powered PC running *SuSE Linux 8.0*. Four of thirteen slaves processes ran on an *Intel PIV 1.7GHz* powered PC, the other nine on an *Intel PIV 1.8 GHz*, all running *SuSE Linux 8.0*. The slaves for an experiment are chosen at random from the pool of available machines to avoid influences of these minor differences on the results. Every participating PC is configured with *Java SDK 1.4.1* and *Jini 1.2.1*. The PC's are connected using a 100Mbps full switched ethernet network.

5.2. Results

The total number of tests effectively executed using the CCC design can be derived from Table 1 (where: $m = 5$, $n = 3$ and $m_c = 10$), resulting in $5 * (2^3 + 2 * 3) + 10 = 80$ tests. For each test the total round trip time T_{JSi}, with $1 \leq i \leq 80$, is measured. For each T_{JSi}, the speedup S_{JSi} can be computed using (2).

5.2.1. Total Time T_{JS} Table 2 shows the ANOVA table for the outputfactor T_{JS}. Only the (combinations of) factors that have significant influence on the outputfactor are summarised. First of all, one can remark that all (combinations of) inputfactors, except for B^2 and C^2, seem to have significant influence on the outputfactor T_{JS}. This can be seen by the fact that the P-value in the last column is smaller than the LoS which was set to five percent. From this table the value of the R^2 statistic can be calculated, which is a measure for the accuracy of the fitted model [13]. The closer to 1, the better the fit. Values of 0.9 or above are considered to be good. The resulting R^2 statistic for the model is 0.9788, which is very good. This means that the model can correctly explain over 95% of the variance.

2 Remark that the inputfactors must be integer values.

Effect	MS	F-value	P-value
I	$6.333e+09$	46.24	< 0.0001
A	$1.053e+10$	76.88	< 0.0001
B	$1.442e+10$	105.26	< 0.0001
C	$1.264e+10$	92.29	< 0.0001
A^2	$1.624e+09$	11.86	0.0108
AB	$1.320e+09$	9.64	0.0172
AC	$1.237e+09$	9.03	0.0198
BC	$2.565e+09$	18.73	0.0034
E	$1.370e+08$		

Table 2. ANOVA output for the outputfactor T_{JS} with a CCC design (A: number of slaves; B: number of tasks; C: execution time of one task in ms)

Applying Regression Analysis on the data and the set of factors selected by ANOVA, results in a multivariate Response Surface Model (RSM) for the total time T_{JS}:

$$\begin{aligned} T_{JS} \; = \; & 35419.8 - 15148.9\,A + 95.9668\,B \\ & +86.9889\,C + 1336.16\,A^2 \\ & -12.1637\,AB - 11.7758\,AC \\ & +0.144511\,BC \end{aligned} \qquad (3)$$

This RSM is shown in Fig. 3.

As the model is based on a sparse set of data points, the accuracy deteriorates towards the boundaries of the design space. In this model the effect of factor C is almost negligible if $A = 11$ and $B = 296$, and so T_{JS} is almost reduced to a constant. This can also be seen in Fig. 3. This means that the model predicts that the total round trip time of a JavaSpaces experiment with 11 slaves and 296 tasks of mean execution time 296ms would run as long as the same experiment but for tasks of mean execution time 1000ms, which is not what one would expect. The real data points confirm this. For $A = 11$ and $B = 296$ the RSM given in (3) reduces to:

$$T_{JS} = 19258.4256 + 0.230356\,C$$

resulting in a total predicted time of approximately 19327ms for $C = 296$, and 19489ms for $C = 1000$. On the other hand, the real data points show execution times of approximately 10000ms and 29000ms respectively. The reason for this deviation lies in the model itself. The accuracy of the RSM is the highest inside the design space, but deteriorates towards the edges of the area.

In Fig. 4 one can see sections of Fig. 3 at $C = 648$. This graph suggests that a minimum of the total execution time

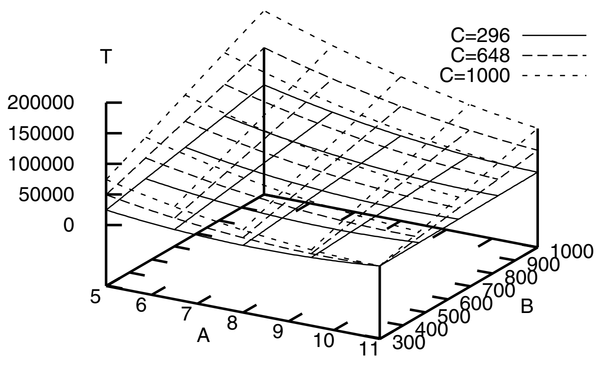

Figure 3. The complete RSM for T_{JS} (A: number of slaves; B: number of tasks; C: execution time of one task in ms)

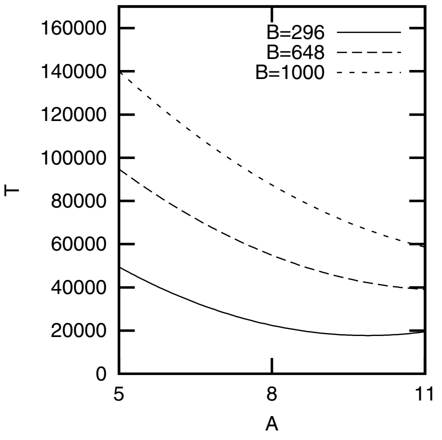

Figure 4. Section of the RSM for T_{JS} at $C = 648$ (A: number of slaves; B: number of tasks; C: execution time of one task in ms)

T_{JS} is found at about 10 slaves. This is mainly caused by the presence of the dominant A^2 factor in the RSM. It is important to note that the purpose of using DoE is to find a subset of significant inputfactors from the total set of inputfactors. From the resulting model, predictions can only be made by interpolation and not by extrapolation. Thus, predictions about data points near the edges of the design space are far less accurate than within the design space. Predictions beyond the edges are completely useless.

5.2.2. Speedup S_{JS} ANOVA analysis for the speedup S_{JS} learns that the factors A, C and C^2 are significant. The resulting R^2 statistic lies above 0.93, which is good. As one could expect, the number of tasks (inputfactor B) has no significant influence. However, the mean execution time for one task (inputfactor C) is important, which means that JavaSpaces is more robust with respect to the number of tasks than to the mean execution time per task. The RSM of the speedup S_{JS} is calculated using Regression Analysis:

$$\begin{aligned} S_{JS} &= -3.50147 + 0.885237\,A \\ &\quad +0.00988414\,C - 0.00000569946\,C^2 \end{aligned} \quad (4)$$

This RSM is shown in Fig. 5.

Equation (4) (and its corresponding figure) show that:

1. larger mean execution times have positive influence on the speedup, and
2. the speedup grows linearly with the number of slaves.

Both observations are obvious, but the exact behaviour (or the numerical coefficients in (4)) will be platform dependent. As communication overhead remains the same for all mean execution times per task, the proportion of both decreases for larger execution times, resulting in increasing speedup.

5.2.3. Residual Histograms One of the conditions to be met in order to apply ANOVA is that the residuals of the outputfactors must be normally distributed [11]. The residual for T_{JS} for test i with parameters a, b and c is defined as

$$R_{Ti} = |T_{JSi} - T_{JS}(a, b, c)|$$

where T_{JSi} is the wallclock time of the ith test and $T_{JS}(a, b, c)$ is the model prediction given by (3). The residuals for S_{JS} are obtained analogously:

$$R_{Si} = |S_{JSi} - S_{JS}(a, b, c)|$$

Both residual histograms are shown in Fig. 6. Neither histograms are perfect, but good enough to justify the application of ANOVA. One of the reasons for the deviation between the residuals and the normal distribution is the relatively small amount of data points (80).

6. Verification of the Experiment

To test the accuracy of the obtained RSM, a number of extra tests were performed. The values for the different inputfactors were chosen in between the factorial points of the

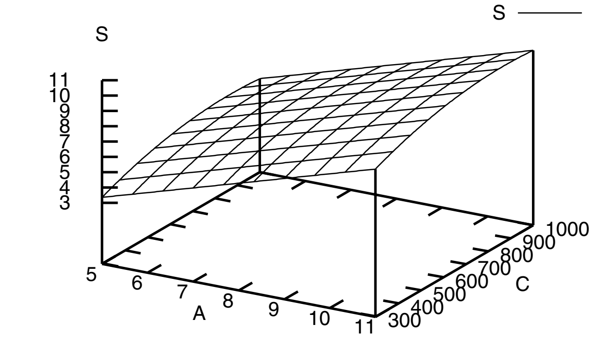

Figure 5. The complete RSM for S_{JS} (A: number of slaves; B: number of tasks; C: execution time of one task in ms)

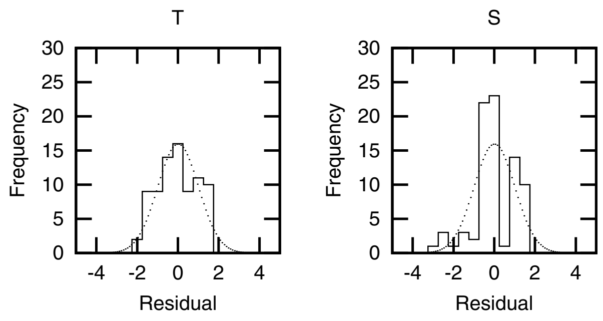

Figure 6. Residual histograms for T_{JS} and S_{JS}

CCC design in order to test the accuracy of the model by interpolation. A total number of 320 extra tests were executed, with 6, 7, 9 and 10 slaves (A), 384, 560, 736 and 912 tasks (B) and 384, 560, 736 and 912 mean execution times (C) per task. The experiment was executed 5 times for all combinations. In the remainder of this section, we will use the mean of the experiment results.

Figure 7 shows both the RSM and the means of the extra tests for the outputfactor T_{JS} for $C = 560$. The mean deviation of actual versus predicted times is less than 3000ms (or 3.3%) for a mean predicted time of 90000ms, which is very good. Again, note that the predictions are less accurate near the edges, but overall the predictions are good.

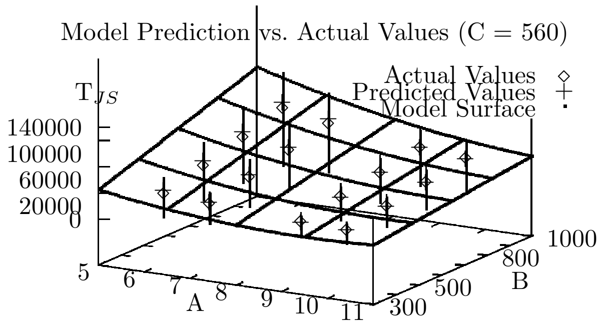

Figure 7. Verification graph for T_{JS} (A: number of slaves; B: number of tasks; C: execution time of one task in ms)

Another way of representing the results of the verification tests is to display the residuals, as shown in Fig. 8 for the outputfactor speedup S_{JS} for $B = 384$. At first sight the variations seem larger than in Fig. 7, but this is of course due to the scale of the Z-axis. The mean of all residuals resulted in -0.03, which is again very good, although the model is again less accurate near the edges, especially for low and high mean execution times (C).

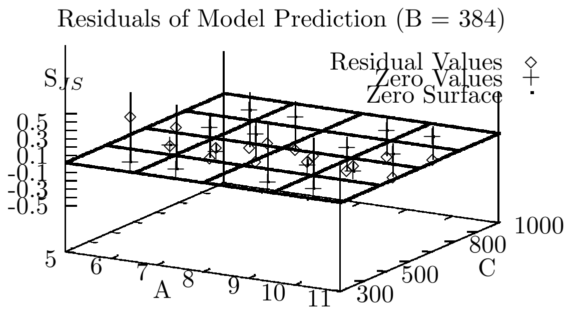

Figure 8. Verification graph of the residuals for S_{JS} (A: number of slaves; B: number of tasks; C: execution time of one task in ms)

7. Conclusions

The main goal of this contribution was to model the performance characteristics of JavaSpaces, a Tuple Space implementation, using a statistical technique called Design of Experiments (DoE). DoE consists of three consecutive steps. In a first step an experiment is constructed, using the Experimental Design (ED) technique, with as few tests as possible, given values of levels for some inputfactors characterising the distributed platform. In the second step the Analysis of Variance (ANOVA) technique is used to select all relevant inputfactors that influence the output. Finally, in step three, Regression Analysis (RA) is performed on these selected inputfactors, resulting in a Response Surface Model (RSM). This RSM provides a multivariate representation of the performance characterisation of JavaSpaces.

Additional experiments on JavaSpaces have been performed in order to verify the models obtained from DoE. These results were satisfying. The model accuracy deteriorates towards the edges of the design space and is not suited for extrapolation.

Finally, we recall that the overall goal of DoE is to statistically identify a subset of significant inputfactors from a (large) set of inputfactors. The main trends are correctly modelled by the RSM and can be used for optimisation purposes. Results of additional tests can be used to refine the RSM.

References

[1] D. H. Bailey, J. M. Borwein, P. B. Borwein, and S. Plouffe. The quest for Pi. *Mathematical Intelligencer*, 19(1):50–57, January 1997.

[2] M. E. Crovella. *Performance Prediction and Tuning of Parallel Programs*. PhD thesis, University of Rochester, 1994.

[3] H. De Neve. Performantieanalyse van gedistribueerde systemen m.b.v. design of experiments. Master's thesis, University of Antwerp, 2003.

[4] H. De Neve, F. Hancke, T. Dhaene, J. Broeckhove, and F. Arickx. On the use of DoE for characterization of JavaSpaces. In *Proceedings ESM 2003*, pages 24–29. Eurosis, 2003.

[5] W. K. Edwards. *Core Jini*. Prentice Hall, second edition, 2001.

[6] R. A. Fisher. *The Design of Experiments*. Hafner Publishing Company, 1971.

[7] E. Freeman, S. Hupfer, and K. Arnold. *JavaSpaces Principles, Patterns, and Practice*. Addison Wesley, 1999.

[8] D. Gelernter. Generative communication in Linda. *ACM TOPLAS*, 7(1):80–112, January 1985.

[9] F. Hancke, G. Stuer, D. Dewolfs, J. Broeckhove, F. Arickx, and T. Dhaene. Modelling overhead in JavaSpaces. In *Proceedings Euromedia 2003*, pages 77–81. Eurosis, 2003.

[10] C. R. Hicks and K. V. J. Turner. *Fundamental Concepts in the Design of Experiments*. Oxford University Press, Inc., 1999.

[11] D. C. Montgomery. *Design and Analysis of Experiments*. John Wiley and Sons Inc., New York, 2001.

[12] J. Narem. An informal operational semantics of c-linda v2.3.5. Technical report, Yale University, December 1990.

[13] J. Neter, M. H. Kutner, C. J. Nachtsheim, and W. Wasserman. *Applied Linear Statistical Models*. WCB/McGraw-Hill, fourth edition, 1996.

[14] M. S. Noble and S. Zlateva. Scientifi c computation with JavaSpaces. Technical report, Harvard-Smithsonian Center For Astrophysics and Boston University, 2001.

[15] M. S. Phadke. *Quality Engineering Using Robust Design*. US Imports & PHIPEs, 1989.

[16] R. K. Roy. *Design of Experiments using the Taguchi Approach*. John Wiley & Sons, Inc., 2001.

[17] J. M. Schopf. *Performance Prediction and Scheduling for Parallel Applications on Multi-User Clusters*. PhD thesis, University of California, 1998.

[18] I. Sun Microsystems. Java rmi: A new approach to distributed computing. Technical report, Sun Microsystems, Inc., 1999.

[19] I. Sun Microsystems. JavaSpaces service specifi cation 1.2. Technical report, Sun Microsystems, Inc., 2001.

[20] G. Sutcliffe, J. Pinakis, and N. Lewins. Prolog-linda: An embedding of linda in muprolog. Technical report, University of Western Australia, 1989.

[21] P. N. K. Timothy W. Simpson, Jesse D. Peplinski and J. K. Allen. On the use of statistics in design and the implications for deterministic computer experiments. Technical report, Woodruff school of Engineering, Atlanta, Georgia, USA, September 1997.