Integrating performance modeling in the software development process

Simonetta Balsamo and Marta Simeoni

Dipartimento di Informatica, Università Cà Foscari di Venezia,
balsamo,simeoni@dsi.unive.it

Abstract. We discuss the integration of performance modeling and analysis in the software development process. Various approaches have been recently defined to integrate performance models and specification languages and models to derive or validate non-functional properties of the software system. Such integration of quantitative performance analysis should provide feedback easily understandable by the software designer and system developers. A framework that allows the combination of different performance modeling techniques and methods, defined at different levels of abstraction, should better support performance analysis and validation of complex and heterogeneous software systems during the software development process.

1 Introduction

Performance analysis of software systems is a critical issue in the software development process. Modeling is an important and useful approach for performance evaluation and system validation and it can provide prediction and comparison of design alternatives.

Software performance engineering deals with the representation and analysis of the software system dynamic based on performance models to provide feedback in the software development process. Several approaches have been defined in the last decade to integrate performance models and specification languages and models to derive or validate non-functional properties of the software system [32, 34, 35].

Most of these integrated approaches are based on various formal specification models ranging form process algebra and Petri net models as well as more recent modeling languages such as UML, and they derive performance evaluation models based on different formalism, such as queueing networks, stochastic process algebras, stochastic Petri nets and Markov processes.

An important issue of the integration of quantitative performance analysis in the software process is the ability to provide feedback that can be easily interpreted by the software designer and system developers.

Since the different performance models can be applied to represent the software system at different levels of abstraction and have different modeling constraints and solution methods, one can take advantage of the relative merit of

each of them. We discuss how the definition of a framework that allows the combination of different performance modeling techniques and methods should better support performance analysis and validation of complex and heterogeneous software systems, at different levels of abstraction during the software development process. Specific software system characteristics such as heterogeneity, scalability and mobility in distributed system could be considered and modeled for the software performance analysis.

2 From software specification to performance model

The Software Performance Engineering (SPE) methodology introduced by Smith in [32] has been the first comprehensive approach to the integration of performance analysis into the software development process, from the earliest stages to the end. More recent approaches focus on the derivation of a performance model right from the Software Architecture (SA) specification [2, 6, 33, 10]. In fact, in the last years SAs have been devised as the appropriate design level to conduct performance analysis, thus allowing for a choice among alternative architectures on the basis of quantitative aspects [31].

In this section we discuss some recent proposals to the derivation of a performance model from the specification of a software system, with the aim of giving the idea of the state-of-the-art in this research area. We proceed by briefly recalling the commonly used specification languages and, in particular, how they are used for the purpose of deriving performance models. We describe then the main classes of performance models and, finally, we present some integrated approaches to the derivation of performance models from software system specifications.

2.1 Specification languages

The recent proposals on the derivation of performance models from SA specifications refer to various specification languages, ranging from the Unified Modeling Language (UML) [9] to formal specification languages like process algebras and Petri nets, and other specification techniques embedded into commercial (object oriented) software design environments. The choice of UML is motivated by the fact that UML is nowadays a widely used notation for the specification of software systems. It provides various diagrams allowing the description of the different system views, which capture static and behavioral aspects, interactions among system components, and implementation details. On the other hand, the choice of a formal specification language based on Process Algebras or Petri nets is motivated by the possibility of integrating functional and performance aspects and analysis into a unique formalism.

The approaches based on UML as specification language usually introduce additional information to the UML diagrams in order to complete the software specification for the derivation of a performance model ready to be evaluated. Such additional information is associated to the various diagrams by means of

simple annotations or extensions based on stereotypes and tagged values. The approaches based on process algebras and Petri nets usually refer to stochastic extensions of such formalisms, which allow action or firing duration to be expressed by means of random variables.

2.2 Performance models

The main classes of performance models are queueing networks (QNs) [20, 1] and their extensions (namely, extended queueing networks and layered queueing networks [30]), stochastic timed Petri nets (STPNs) and stochastic process algebras (SPAs). Each of these classes has its own peculiarities in terms of expressiveness (i.e. the kind of systems that can be suitably modeled), level of abstraction (i.e. the degree of detail needed to correctly model the system components), and efficiency of the solution methods. However, queueing networks are traditionally the most commonly used. Their popularity is due to a relative high accuracy in the performance results and the efficiency in model analysis and evaluation.

A performance model of the above classes can be analyzed by analytical methods or by simulation, in order to evaluate a set of performance indices such as resource utilization, throughput, customer response time and others. Simulation is a widely used general technique whose advantage is the modeling flexibility, but whose main drawback is the potential high development and computational cost to obtain accurate results. On the other hand, analytical methods require that the model satisfies a set of assumptions and constraints, and are based on a set of mathematical relationships that characterize the system behavior.

The analytical solution is often derived by considering an underlying stochastic process, usually a discrete-space continuous-time homogeneous Markov chain (MC). The solution of the associated MC is in general numerically expensive because its state space grows exponentially with the number of components of the performance model. However, in some case, some efficient analytical algorithms have been defined for performance models that satisfy some given constraints.

Besides being a possible solution technique for the introduced performance models, simulation can be considered as a performance evaluation technique by itself. Existing simulation tools provide a suitable specification language for the definition of simulation models, and a simulation environment to conduct software system performance evaluation.

2.3 Some integrated approaches

We discuss now some recent approaches to derive performance models from software system specifications. We take into consideration just a few main methodologies, with the aim of giving the idea of the state-of-the-art in this research area. In describing the various methodologies we focus on the system specification phase and, in particular, on the effort and the specific performance evaluation knowledge requested to the system designer and developer in order to apply the transformation methodology.

Some of the methods proposed in the literature [36, 11, 28, 27, 15] refer to the SPE methodology [32], which is based on two models: the software execution model and the system execution model. The former is based on execution graphs and represents the software execution behavior; the latter is based on QN models and represents the system platform, including hardware and software components. The analysis of the software model gives information concerning the resource requirements of the software system. The obtained results, together with information about the hardware devices, are the input parameters of the system execution model, which represents the model of the whole software/hardware system.

Among the approaches based on SPE we consider the one developed by Cortellessa and Mirandola in [11]. They propose the PRIMA-UML methodology, which makes a joint use of information from different UML diagrams to incrementally generate a performance model representing the specified system. SAs are specified by using Deployment, Sequence, and Use Case diagrams. The software execution model is derived from the Use Case and Sequence diagrams, and the system execution model from the Deployment diagram. Moreover, the Deployment diagram allows the tailoring of the software model w.r.t information concerning the overhead delay due to the communication between software components. Both Use Case and Deployment diagrams are enriched with performance annotations concerning workload distribution and parameters of devices, respectively. Hence, in order to apply the PRIMA-UML methodology, the software designer is required to have a specific knowledge on these data and on how to specify them. In [14] the methodology has been extended in order to cope with the case of mobile SAs: starting from a SA described in UML notation (extended to better modeling the possible adoption of mobility-based paradigms), the PRIMAmob-UML approach generates the corresponding software and system execution models allowing the designer to evaluate the convenience of introducing logical mobility into a software application. The authors define extensions of execution graphs and extended QNs to model the uncertainty about the possible adoption of code mobility.

Other main proposals based on the SPE methodology are the ones developed by Petriu and co-authors in [28, 27, 15]. They propose three conceptually similar approaches where SAs are described by means of architectural patterns (such as pipe and filters, client/server, broker, layers, critical section and master-slave) whose structure is specified by UML-Collaborations (which are similar to Class diagrams showing explicitly the collaborating objects) and whose behavior is described by Sequence or Activity diagrams. The three approaches propose systematic methods based on graph transformation techniques to build layered queueing network models out of complex SAs based on combinations of the considered patterns.

Beside the SPE based approaches there are other conceptually different proposals [2, 33, 21, 29] which still derive QN based performance models. In particular, two approaches described in [2, 33] are based on the generation and analysis of traces (sequence of events/actions) gathered from scenarios describing the be-

havior of the software system under consideration: the more complete one has been developed by Woodside et al. [33], and propose the automatic derivation of a layered queueing network model from a commercial software design environment called ObjecTime Developer [24], by means of an intermediate prototype tool called PAMB (Performance Analysis Model Builder). The domain of application of the methodology is the real-time interactive software, and it applies to the whole development cycle, from the design stage to the final product.

ObjecTime Developer allows the designer to describe a set of communicating actor processes, each controlled by a state machine, plus data objects and protocols for communications. It is possible to "execute" the design over a scenario by inserting events, stepping through the state machines, and executing the defined actions. Moreover, the tool can generate code from the system design, that can be used as prototype or as a first version of the product. The approach in [33] takes advantage of such code generating and executing scenarios capabilities for model-building: the prototype tool PAMB, integrated with ObjecTime Developer, keeps track of the execution traces, and captures the resource demands obtained by executing the generated code in different execution platforms. Essentially, the trace analysis allows the building of the various layered queueing network submodels (one for each scenario) which are then merged into a global model, while the resource demands data provides the model parameters. After solving the model through an associated model solver, the PAMB environment reports the performance results by means of performance annotated Message Sequence Charts and Graphs of predictions. Note that this approach requires less knowledge about performance aspects to the software designer, since the performance parameters are automatically evaluated by the PAMB tool.

Some authors propose the translation of UML models into stochastic Petri net models or stochastic process algebra specifications [25, 22, 7] in order to perform quantitative evaluation of software system. However, the more mature approaches based on formal specification languages such as Petri nets and process algebras do not consider external specification techniques. All of them are instead motivated by the possibility of integrating functional and non-functional aspects and providing a unique reference framework and model for SA specification and performance. However, from the performance evaluation viewpoint, the analysis usually refers to the numerical solution of the underlying Markov chain which can easily lead to numerical problems due to the state space explosion. Moreover, another disadvantage of these approaches is that the software designer is required to be able to specify the software system using process algebras or Petri nets, and to put the correct parameters to action or firing duration.

In particular, several stochastic extensions of process algebras have been proposed in the last years, such as TIPP (TIme Processes and Performability evaluation) [13], EMPA (Extended Markovian Process Algebra) [8, 3] and PEPA (Performance Evaluation Process Algebra) [18, 19]. The main differences between these SPAs concern the the expressivity of their languages. All of them associate exponentially distributed random variables to actions, and provide the

generation of a Markov chain out of the semantic model (a labeled transition system enriched with time information) of a specified system. Furthermore, all of them are supported by appropriate tools (the TIPP tool, PEPA Workbench and Two Towers —based on EMPA).

In this setting, Balsamo et alt. introduced in [4] an SPA based Architectural Description Language (ADL) called Æmilia (an earlier version called ÆMPA can be found in [5]), whose semantics is given in terms of EMPA specifications. The introduction of an ADL ease the software designer in identifying the system components and their interconnections: Æmilia provides in fact a formal framework for the compositional, graphical, and hierarchical modeling of software systems, and is equipped with some functional checks for the detection of possible architectural mismatches. The authors propose moreover a systematic approach to the translation of Æmilia specifications into QN models, with the aim of taking advantages of the orthogonal strengths of the two formalisms: formal techniques for the verification of functional properties for Æmilia (SPAs in general), and efficient performance analysis for QNs.

The last approach we consider has been proposed by De Miguel et alt. in [12], and is based on simulation models. It uses simulation packages in order to define a simulation model, whose structure and input parameters are derived from the UML diagrams. The approach focus on real time systems, and proposes extensions of UML diagrams to express temporal requirements and resource usage. The extension is based on the use of stereotypes, tagged values and stereotyped constraints. SAs are specified using the extended UML diagrams without restrictions on the type of diagrams to be used. Again, the software designer is required to be aware on how to specify the timing and performance parameters in order to use the whole framework. The resulting diagrams are then used as input for the automatic generation of the corresponding simulation models. They also define a middleware model for scheduling analysis. The simulation model is defined by instantiating with the application information the generic models that represent the various UML metaclasses. The approach generates submodels for each application element and combines them into a unique simulation model of the UML application. The approach provides also a feedback mechanism: after the model has been analyzed and simulated, some results are included in the tagged values of the original UML diagrams.

We point out that most of the existing methodologies still do not provide a complete framework for the automatic derivation of performance models from software specification. Most of them still do not integrate the various steps concerning specification, model transformation, performance evaluation and feedback to the software designer into a unique environment properly supported by tools. In particular, just a few approaches address the issue of reporting (or interpreting) the performance results at the SA level. This, however, is an important feature which has to be considered in order to make the performance evaluation of SA really complete and usable.

3 Open problems and perspectives

Performance modeling and analysis in the software development process is based on the generation of appropriate performance models that allow high integration with functional and behavioral models.

For each class of performance models, from queueing networks to stochastic process algebras, stochastic Petri net and Markovian processes, we can identify specific advantages, constraints and peculiarities in terms of expressiveness, efficiency of the solution methods and level of abstraction.

Therefore we can take advantage of the characteristics of the modeling formalisms to define a combined or integrated framework for software performance analysis, by combining different types of models. Such framework should better support performance analysis and validation of complex and heterogeneous software systems, at different levels of abstraction during the software development process. By referring to different levels of abstraction in the development process, one can identify and select the appropriate performance model that can be applied to evaluate the software system. This corresponds to model the appropriate features and details of the software system that allow to describe its dynamic behavior.

This performance modeling approach is illustrated in Fig. 1. Starting from an appropriate software specification, from the description of its dynamic behavior one can choose and derive a performance model that allows representing the relevant characteristics at the selected level of abstraction. By applying the corresponding transformation algorithm a performance model is defined, selecting from analytical model, such as queueing networks, Markov chains, stochastic Petri nets and stochastic process algebras, simulation model and possibly other types of models such as hybrid models.

Once the appropriate performance model has been identified, we can apply the corresponding performance evaluation methods. When an analytical performance model is selected, solution methods can be symbolic for some particular classes, such as some simple Markov chains and the class of product-form queueing networks [20], so allowing an easier parametric analysis and result interpretation at the software level. More generally, analytical models can be analyzed through approximate numerical algorithms, such as for extended queuing networks and layered queueing networks [30], and for more complex Markov chains. Stochastic Petri nets and Stochastic Process Algebras are usually analyzed by defining and numerically solving the underlying Markov chain. Simulation models are analyzed by discrete simulation techniques. The analysis and solution of the performance model requires the instantiation of system parameters, whose type and number depends on the level of abstraction of the model. Parameter instantiation can be determined by a scenario driven approach.

Hence, the selection of the performance solution method depends on the selected performance models and parameters and can provide different degrees of accuracy of the performance evaluation figures of merit.

This performance modeling framework should provide the designer a feedback that has to be easily interpreted by the software designer without specific

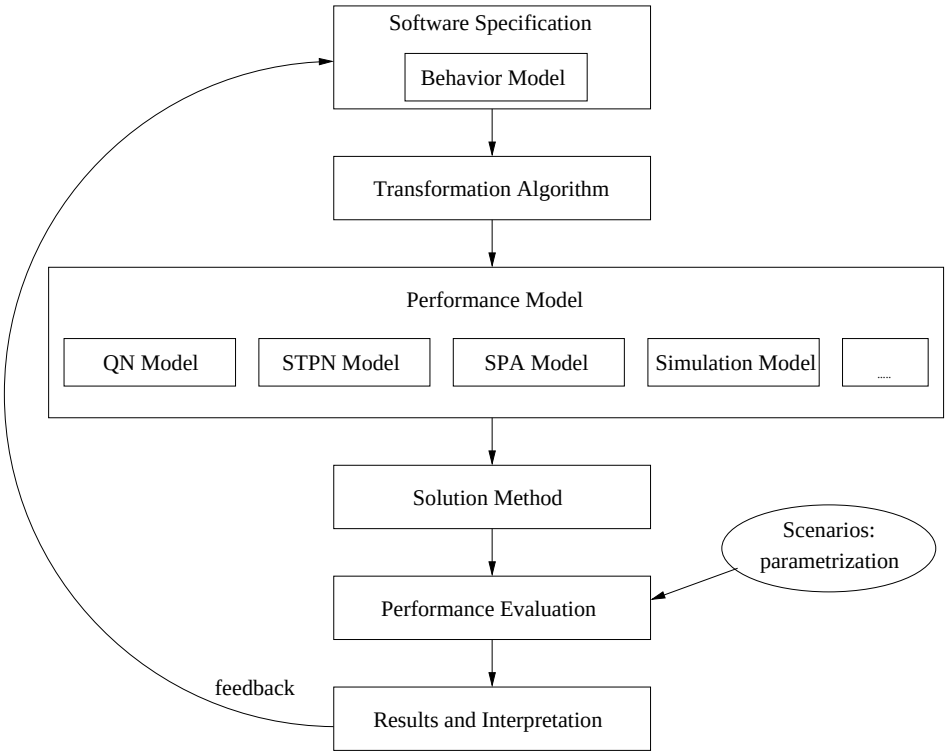

Fig. 1. Performance modeling of software specification.

knowledge of performance modeling techniques. This feature strongly depends on the selected performance modeling approach.

Hence the selection of the appropriate model should be provided by the performance environment and driven by several issues, as discussed in the previous section. These include the software specification model or language, the relevant characteristics of the software systems, the accuracy of the performance analysis required in the software development step and the ability to provide feedback at the software designer.

Specific software system characteristics such as heterogeneity, scalability and mobility in distributed system could be considered and represented by the appropriate model for the software performance analysis.

An integrated software performance environment should allow successive refinements of performance modeling based on various models, corresponding to different levels of abstraction, to be applied at different phases of the software development process, driven by the specified requirements. Then the scheme illustrated in Fig. 1 can be iterated by considering different performance models.

Additional information should be possibly integrated in successive steps to define either different or more refined performance models that allow further software system evaluation. Therefore in this framework it is relevant to investigate the application of methodologies to derive structured performance models at different levels of abstraction, where the solution of a model at a given level can be obtained extending the solution of a related model at the previous level. These methodologies are based on the decomposition and aggregation principle and can be applied to various classes of performance models, from Markov chain to queueing networks, to stochastic Petri nets and stochastic process algebras [20]. In this direction an interesting tool for the analysis of complex systems with models with a high level of details is given by hybrid modeling that integrate simulation and analytical modeling approaches.

Another relevant issue in this framework of software performance is to identify and take advantage of the specific characteristics of the various classes of performance models, possibly integrating various models into the same framework. Starting from the methodologies defined to derive a class of performance model from the software specification, some examples of integration of software performance approaches have been recently presented. The comparison of layered queueing networks and stochastic process algebras and their application to performance validation is described in [16]. The combination of stochastic process algebras and basic queueing network models for software architecture analysis introduced in [4] to take advantage of formal techniques to verify functional properties for the former model and efficient solution algorithms for the latter. An automated software design performance environment has been presented in [33], and a performance approach that considers software in a distribute mobile environment has been recently introduced in [10]. Further research has to investigate a more complete integration of different models and solution techniques, including simulation and hybrid methods, into the software performance process, exploiting the identification of appropriate and representative models and the problem of how to efficiently feedback the performance results at the software specification level.

Acknowledgments

We thank Antinisca di Marco for useful comments and suggestions about the translation methodologies described in this paper.

References

1. M. Ajmone, G. Balbo, G. Conte "Performance Models of Multiprocessor Performance". The MIT Press (1986)
2. F. Aquilani, S. Balsamo, P. Inverardi "Performance Analysis at the Software Architectural Design Level". *Perfomance Evaluation* Vol.45, Issue 2-3, pp. 147–178 (2001)
3. M. Bernardo "Theory and Application of Extended Markovian Process Algebra". Ph.D. Thesis, University of Bologna, Italy (1999)

4. S. Balsamo, M. Bernardo, M. Simeoni "Combining Stochastic Process Algebras and Queueing Networks for Software Architecture Analysis". In [35] (2002)
5. M. Bernardo, P. Ciancarini, L. Donatiello "ÆMPA: A Process Algebraic Description Language for the Performance Analysis of Software Architectures". In [34], pp. 1–11 (2000)
6. M. Bernardo, L. Donatiello, P. Ciancarini "Stochastic Process Algebra: From an Algebraic Formalism to an Architectural Description Language". To appear in *Performance Evaluation of Complex Systems: Techniques and Tools*, Springer LNCS (2002)
7. S. Bernardi, S. Donatelli, J. Meseguer "From UML Sequence Diagrams and Statecharts to analysable Petri Net models". In [35] (2002)
8. M. Bernardo, R. Gorrieri "A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time". *Theoretical Computer Science* Vol. 202, pp. 1–54 (1998)
9. G. Booch, J. Rumbaugh, I. Jacobson "The Unified Modeling Language User Guide". Addison Wesley, New York (1999)
10. V. Cortellessa, V. Grassi "A performance based methodology to early evaluate the effectiveness of mobile software architectures". *Journal of Logic and Algebraic Programming* (2002)
11. V. Cortellessa, R. Mirandola "Deriving a Queueing Network based Performance Model from UML Diagrams". In [34], pp. 58–70 (2000)
12. M. De Miguel, T. Lambolais, M. Hannouz, S. Betgé-Brezetz, S. Piekarec "UML Extensions for the Specification and Evaluation of Latency Constraints in Architectural Models". In [34], pp. 83–88 (2000)
13. N. Götz, U. Herzog, M. Rettelback "TIPP—a language for timed processes and performance evaluation". *Technical Report* 4/92, IMMD7, University of Erlangen-Nürnberg, Germany (1992)
14. V. Grassi, R. Mirandola "PRIMAmob-UML: A Methodology for Performance Analysis of Mobile Software Architectures". In [35] (2002)
15. G. Gu, Dorina C. Petriu "XSLT transformation from UML models to LQN performance models". In [35] (2002)
16. U. Herzog, J. Rolia "Performance Validation tools for software/hardware systems". *Perfomance Evaluation*, Vol. 45, Issue 2-3, pp. 125–146 (2001)
17. J. Hillston "A Compositional Approach to Performance Modelling". Cambridge University Press (1996)
18. J. Hillston "PEPA - Performance Enhanced Process Algebra". *Technical Report*, Dept. of Computer Science, University of Edimburgh (1994)
19. J. Hillston "A Compositional Approach to Performance Modelling". Cambridge University Press, Distinguished Dissertation Series (1996)
20. K. Kant "Introduction to Computer System Performance Evaluation". *McGraw-Hill* (1992)
21. P. Kähkipuro "UML-based Performance Modeling Framework for Component-Based Distributed Systems". *Proc. of Performance Engineering*, Springer LNCS 2047, pp. 167–184 (2001)
22. P. King, R. Pooley "Derivation of Petri Net Performance Models from UML Specifications of Communication Software". *Proc. of XV UK Performance Engineering Workshop* (1999)
23. D. Menascè, H. Gooma "A Method for design and Performance Modeling of Client/Server Systems". *IEEE Trans. on Software Engineering*, Vol. 26, n. 11, pp. 1066–1085 (2000)

24. ObjecTime Ltd., Developer 5.1 Reference Manual, ObjecTime Ltd., Ottawa, Ont. (1998)
25. R. Pooley "Using UML to Derive Stochastic Process Algebra Models". *Proc. of XV UK Performance Engineering Workshop* (1999)
26. R. Pooley, P. King "The Unified Modeling Language and Performance Engineering". In *Proc. IEEE Software* (1999)
27. Dorina C. Petriu, H. Shen "Applying UML Performance Profile: Graph Grammar-Based Derivation of LQN Models from UML Specifications". *Proc. of TOOLS02, Springer Verlag LNCS 2324*, pp. 159–177 (2002)
28. Dorina C. Petriu, X. Wang "From UML descriptions of High-Level Software Architectures to LQN Performance Models". *Proc. of AGTIVE'99, Springer Verlag LNCS 1779*, pp. 47–62 (1999)
29. Dorin C. Petriu, M. Woodside "Software Performance Models from System Scenarios in Use Case Maps". *Proc. of TOOLS02, Springer Verlag LNCS 2324*, pp. 141–1158 (2002)
30. J.A. Rolia and K.C. Sevcik "The Method of Layers". *IEEE Transaction on Software Engineering*, Vol. 21, n.8, pp. 682–688 (1995)
31. M. Shaw, D. Garlan "Software Architecture: Perspectives on an Emerging Discipline". Prentice Hall (1996)
32. C. Smith "Performance Engineering of Software Systems". Addison-Wesley, Reading, MA (1990)
33. M. Woodside, C. Hrischuk, B. Selic, S. Bayarov "Automated performance modeling of software generated by a design environment". *Performance Evaluation*, Vol. 45, Issue 2-3, pp. 107–123 (2001)
34. Proceedings of the 2nd Int. Workshop on Software and Performance (WOSP 2000), ACM Press, Ottawa, Canada (2000)
35. Proceedings of the 3rd Int. Workshop on Software and Performance (WOSP 2002), ACM Press, Rome, Italy (2002)
36. L.G. Williams, C.U. Smith "Performance Evaluation of Software Architectures". *Proc. of WOSP'98, Santa Fe, New Mexico, USA*, pp. 164–177 (1998)