# Online Model-Based Adaptation for Optimizing Performance and Dependability*

Kaustubh R. Joshi‡, † Matti Hiltunen§, Richard Schlichting§, William H. Sanders‡, and Adnan Agbaria‡

‡Coordinated Science Lab
University of Illinois at Urbana-Champaign
Urbana, IL, USA
{joshi1,whs,adnan}@crhc.uiuc.edu

§AT&T Labs Research
180 Park Ave.
Florham Park, NJ, USA
{hiltunen,rick}@research.att.com

## ABSTRACT

Constructing adaptive software that is capable of changing behavior at runtime is a challenging software engineering problem. However, the problem of determining when and how such a system should adapt, i.e., the system's *adaptation policy*, can be even more challenging. To optimize the behavior of a system over its lifetime, the policy must often take into account not only the current system state, but also the anticipated future behavior of the system. This paper presents a systematic approach based on using Markov Decision Processes to model the system and to generate optimal adaptation policies for it. In our approach, we update the model on-line based on system measurements and generate updated adaptation policies at runtime when necessary. We present the general approach and then outline its application to a distributed message dissemination system based on AT&T's iMobile platform.

## 1. INTRODUCTION

The goal of self-managing system design is to allow a system to maintain or maximize its performance in spite of changing environmental conditions. To achieve this goal, it is necessary to answer some fundamental questions about how a software system can adapt its behavior, when the system should adapt, and which of the possibly many adaptation actions it should choose. Many researchers have focused on providing *mechanisms* for building adaptive software (e.g., [9, 14]), while using simple heuristics or adhoc rules as the adaptation *policies* that dictate how the mechanisms are employed. However, adhoc policy design may not always be reasonable, especially when the adaptations involve trade-offs, and the choice of the correct adaptation decision is not immediately obvious. In such situations, more quantitative approaches to policy generation are desirable.

Recently, work has been done on generating adaptation decisions systematically using techniques from linear feedback control theory [8, 2, 10]. However, those techniques have been successfully applied only to performance management of single-server systems with varying workload as the only component of environmental change, and seem suitable only when the performance criterion is expressible as a very simple (typically linear) closed form function of the system's observables and control inputs. Moreover, the techniques are primarily reactive in that they constantly monitor the current levels of the performance metric to be controlled, and inject corrective actions if the performance metric deviates from the required value. However, when the problem is generalized to systems with multiple components, not only are linear formulations of system properties difficult to formulate, but component failures become a non-trivial component of environmental change that must be considered. However, component failures introduce many difficulties that make the use of previously considered techniques unsuitable. For example, failures are rare events, and often have effects that are far more serious than performance tuning changes. Therefore, a "measure and correct" approach does not work for failures. Moreover, dependability-oriented metrics are usually defined over long intervals of time. Therefore, any control scheme for optimizing them must not only be predictive (in the sense that it must take into account future behavior), but be so over long periods of system operation.

In this paper, we propose a general approach that attempts to tackle the above challenges, and we apply it to an operational enterprise messaging system. The approach uses Markov decision theory to generate adaptation policies that optimize combined performance and dependability measures in the face of changing environmental conditions. The use of Markov decision theory has been proposed as a mathematical framework for making resource management and adaptation decisions in [12, 7, 1]. However, there are

*This material is based on work supported by the National Science Foundation under Grant No. CNS-0406351. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. The first author is also supported in part by an AT&T Virtual University Research Initiative (VURI) fellowship.

†The work was done while this author was at AT&T Labs Research.

To appear in the Workshop on Self-Managing Systems (WOSS'04), Oct. 2004, Newport Beach, CA.

important differences between our work and past efforts.

The biggest difference is that none of the previous work explicitly considers the problem of large state spaces needed in real operational systems such as the one presented in this paper. One of the primary goals of our work is to explore techniques that allow large state spaces to be handled more efficiently by taking advantage of temporal properties (exploited by using online solutions) and model structure. The specific techniques we propose include aggregation and time-scale decomposition. Time-scale decomposition is possible because we explicitly assume on-line solution of certain parts of the model, whereas previous work usually assumed that the models were static. Additionally, where previous work focused on determining the optimal time to summon a human repairman ([6] and [12]), we completely remove all human repair processes from the model. This is in keeping with the self-managing system goal of assuming no human intervention, and leads to models that are more efficient to solve.

There are also differences between our work and past work in the assumptions made and the algorithms used. For example, most previous work has dealt with discrete-time systems ([12, 7, 1]), which makes it hard to model failures that can occur at any point in time. The only previous continuous-time work we are aware of ([6]) deals with systems with a known lifetime. That assumption seems reasonable only for mission-oriented military and space systems.

Finally, our approach is model-based, and relies on the system designer to explicitly specify a model of the system behavior. In that respect, it differs from Reinforcement-Learning (RL) [13], which is also based on Markov decision theory, but does not assume prior knowledge of the model. Therefore, RL requires (possibly long) periods of exploration during which optimal actions are not taken, but the controller tries to learn the system dynamics and rewards. Although model-free approaches are needed when not enough is known about the system dynamics or rewards, that is not the case in the domain of dependability and performance of computer systems, for which modeling has a long history.

## 2. PROPOSED APPROACH

In the proposed approach, we add an adaptation controller component to the software system under consideration. The controller maintains an internal model of the system being controlled. This internal model, and an initial set of numerical values that quantify the model, are specified by the system designer. Runtime instrumentation takes measurements of the system while it is operational, and passes them to the controller. The controller uses the measurements to update and re-solve the model periodically. The model solution algorithms generate the adaptation actions that the controller propagates to the rest of the system. The propagation of actions can be either coordinated [4] or uncoordinated depending on the application. The model solution algorithms are completely deterministic, thus allowing the controller to be replicated for dependability. In the remainder of this section, we describe the mathematical framework that the controller uses to represent applications internally, and the solution algorithms used for generation of adaptation actions.

### 2.1 System Model

Let $M$ be the set of "operating modes" of the system and $W$ be the set of possible states of the system, environment, and resources (i.e., the "world"). Note that the number and type of operating modes depends on how adaptive the software system has been designed to be. For example, the system may adapt by starting up additional replicas of a software component or the software components may themselves be designed to be adaptive, that is, be capable of executing in two or more operating modes. Further, let $E$ be a set of external events that can occur and change the world state and $A$ be a set of adaptation actions that the system can take to change its operating mode. For any combination of world state $w$ and operating mode $m$, only a subset of all possible external events can occur, and only a subset of adaptations can be performed. They are denoted by $E_{w,m}$ and $A_{w,m}$ respectively.

Each *event* $e$ is associated with a mapping $nw_e : W \times M \rightarrow W$ that specifies the new world state that results from the occurrence of the event if the system is in a particular state and mode. In a given world state and operating mode, multiple events may be enabled. Each enabled event is associated with a random timer. Only that event whose timer expires the earliest actually occurs. The random timers facilitate modeling of stochastic future behaviors of the system. For the purposes of this work, we assume that each random timer is negative exponentially distributed[1] with a rate $\lambda_e(w,m) : W \times M \rightarrow \mathbb{R}^+$. That allows us to map the adaptation process into an underlying continuous time Markov decision process (CTMDP) [11].

Each *adaptation action* occurs instantaneously and is associated with a mapping $nm_a : W \times M \rightarrow M$ that specifies the new operating mode the system enters because of the action. At any time, it might be that no adaptation actions are available, or the system might choose not to adapt. Hence, a special action $a_\perp$ is defined for which $nm_{a_\perp}(w,m) = m, \forall w \in W, m \in M$.

After an event has occurred, the controller is given a chance to perform an adaptation action to react to the event. More precisely, if the system starts in state $(w,m)$, some event $e$ in $E_{w,m}$ occurs, and the system adapts by choosing some action $a$ in $A_{w',m}$, the evolution of the system can be described as:

$$(w,m) \overset{e \in E_{w,m}}{\rightarrow} (\overbrace{nw_e(w,m)}^{w'}, m) \overset{a \in A_{w',m}}{\rightarrow} (w', \overbrace{nm_a(w',m)}^{m'})$$

Note that for a system, it may be that not all possible combinations of $W$ and $M$ are reachable from a particular initial state. Hence, the adaptation state space $S$ may be defined as the subset of $W \times M$ that contains only reachable combinations of $W$ and $M$.

There are three distinct types of rewards for each adaptation state-space state $(w,m)$. *Action rewards* ($k_a(w,m)$, $a \in A_{w,m}$) specify the impulse of reward that a process obtains when action $a$ is chosen in state $(w,m)$. *Rate rewards* ($c(w,m)$) specify the rate at which a process receives a reward while in state $w,m$. Finally, *event rewards* ($k_e(w,m)$, $e \in E_{w,m}$) specify an impulse reward that the process obtains when event $e$ occurs in state $(w,m)$. Note that the "reward" may be a negative value or "cost," e.g., if the adaptive

[1]Negative exponential random variables (RVs) are a widely accepted way to represent both times between component failures (particularly hardware failures) and unknown arrival processes (e.g., workloads). Additionally, many commonly used distributions can be approximated by multiple negative exponential RVs using standard techniques.

action requires the system to be brought down temporarily or if an event makes the system in its current mode unable to provide service.

Using the above terminology, an adaptation policy $\pi$ is a mapping $\pi : S \rightarrow A$ that specifies an adaptation action $A_{w,m}$ for every reachable state of the adaptation model.

## 2.2 Optimal Policy Generation

Since our goal is to consider the effects that adaptation actions may have in the future in addition to immediate effects, a description of how rewards are accumulated over time is needed. The *discounted reward criterion* [11] was chosen for this purpose. This criterion computes the total accumulated reward over time, but discounts rewards obtained $t$ time units into the future by the factor $e^{-\alpha t}$, where $\alpha$ is the discount rate. Discounting is important because the model is usually only an approximation of the true system behavior, and therefore its predictive power can diminish with increasing time, making future reward predictions less reliable. More precisely, it can be shown that maximizing the discounted reward is equal to maximizing the total accumulated reward under the condition that the system is operational under the current assumptions only for a negative exponentially distributed (with rate equal to the discount factor) random amount of time. In practice, the discount factor is an empirical parameter set according to the dynamics of the system, and how quickly the model is expected to change.

In order to solve the adaptation model, we use the modified policy-iteration algorithm for discounted discrete-time Markov Decision Processes (DTMDPs) adapted to fit the model formulation described in the previous section. In order to be able to use the algorithm, we first compute the average one-step reward the process obtains on choosing action $a$ in state $(w, m)$ using the action, event, and rate rewards defined above. Then, we convert the continuous time adaptation model into an equivalent discrete time model using the process of *uniformization* [11]. The uniformization process yields the probability that event $e$ occurs in state $(w, m)$ as $p_e(w, m)$, and the equivalent one-step reward obtained by choosing action $a$ in state $(w, m)$ as $r(w, m, a)$.

The modified policy iteration algorithm proceeds by initially choosing some arbitrary policy $\pi$ (i.e., choosing some adaptation action for all combinations of $(w, m)$). Then, the total discounted reward (or "value") obtainable from each state $v(w, m)$ is computed through solution of the set of linear equations:

$$\forall (w,m) \in S, v(w,m) = r(w,m,a) + \\ + \alpha \sum_{e \in E_{w,m'}} p_e(w,m') v(nw(w,m'),m')$$

However, this set of linear equations is expensive to solve when the state space is large. Therefore, we estimate the function $v(w, m)$ by performing a few iterations of an iterative linear equation solver. After experimentation with several iterative linear solution methods, we found that the Gauss-Seidel method with successive over-relaxation (SOR) gave the fastest convergence for the models we used. Therefore, that technique is being used in our work. Once the value function has been estimated, a dynamic programming pass is performed to compute a policy that is *strictly* better than the current policy for all states. We perform a pass by choosing for each state $(w, m)$ the action $a$ that achieves the maximum in the following expression:

$$\max_{a \in A_{w,m}} r(w,m,a) + \alpha \sum_{e \in E_{w,m'}} p_e(w,m') v(nw_e(w,m'),m')$$

The process is then repeated until the new policy generated by the dynamic programming pass is the same as the old one. The resulting adaptation maximizes the expected total discounted reward accumulated during the entire lifetime of the system. This means that adaptation actions that have delayed effects are fully accounted for, thus fulfilling one of our goals as set forth in Section 1.

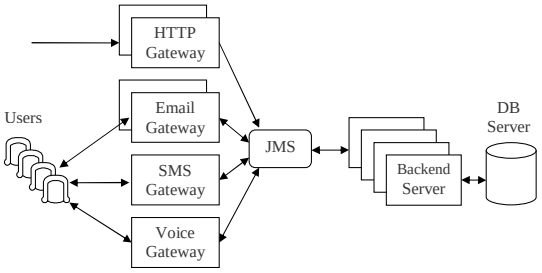


**Figure 1: Messaging System Architecture**

# 3. EXAMPLE: AN ENTERPRISE MESSAGING PLATFORM

In order to demonstrate our approach, we are currently implementing it in the context of a multi-protocol mobile messaging system developed using the AT&T iMobile-EE platform [5]. The system's mission is to provide a *highly available* service for disseminating messages in a *timely fashion*[2] to large groups of end-users through a variety of end-user devices (e.g., PDAs, Blackberry devices, and cellular phones) and protocols (e.g., WAP, SMS, and email). In terms of its dependability and performance goals, the system is representative of a large category of internet-based services that must be highly available, but need only soft real-time guarantees.

## 3.1 System Architecture

The architecture of the messaging system is shown in Figure 1. The system consists of a set of protocol gateways that can communicate with various kinds of devices, and a set of back-end servers that process the messages. The various components communicate with each other using a Java Messaging Server (JMS) provider. Multiple replicas of each component can exist to increase system capacity and provide fault tolerance. The JMS provider performs the required load-balancing functions when forwarding messages. An end-user posts a message using a webpage provided by the HTTP gateway. The HTTP gateway forwards the message to the back-end servers, which look up the database for the preferred contact devices/protocols for each recipient of the message. Next, the outgoing messages are sent to the corresponding protocol gateways for dissemination. The database is used to maintain persistent state about the progress of the message dissemination. When users receive a message, they acknowledge it. The gateways handle the incoming acknowledgments and forward them to the back-

[2]For purposes of this case study, "timely fashion" is specified using a target average message delivery time.

end servers, which then update the message dissemination progress state to indicate successful delivery.

## 3.2 Operational Modes

Two major factors impede the system's ability to meet its goals of highly available and timely message delivery, and we choose operational modes to counter these factors. The factors are system workload and component failure. A high system workload (which is specified in terms of the message arrival rate) can cause a backlog in the system, and impede its ability to deliver messages on time. In order to deal with changing workloads, we introduce four operational submodes that change the manner in which message acknowledgments sent by the user after receiving a message are processed.

The *immediate acks* submode is the normal but expensive submode in which all acknowledgments are received by the gateways, forwarded to the back-end servers, and posted into the database. In the *delayed acks* submode, the acknowledgments are received by the gateways, but not forwarded to the back-end servers for storage in the database. The acknowledgments are stored on the gateways to be sent to the back-end servers at a later time. This submode increases the immediate processing capacity of an overloaded system at the risk of losing the delayed acknowledgments if a failure occurs. The *no acks* submode refuses acknowledgments from the users. Finally, in the *ack transfer* submode, the system transfers the acknowledgments that are stored on the gateways to the back-end servers in addition to performing immediate acknowledgment processing. It is the most expensive resource use mode. To implement these submodes, we will modify the iMobile gateway components to implement these alternative operational modes. We will also extend the gateway components with a control interface that allows the controller to tell the gateway which operational mode to use.

The second factor that impedes the mission of the system is the fact that the hosts on which the messaging system runs are subject to crash failures due to a variety of reasons. For the purposes of our study, we choose the simplest recovery action, which is to simply restart components on hosts that are still operational. Therefore, we introduce a second set of operational submodes that specify a mapping of how the software components that make up the system are distributed on the physical hosts (e.g., HTTP gateway maps to host1 and host3). The set of possible configurations $C_c$ for each component $c$ is the set of sets of hosts on which $c$ can be run, i.e., $2^H$. The set of system configurations is a subset of the Cartesian product of the configuration sets for each component, i.e., $C \subset C_{c_1} \times \ldots \times C_{c_n}$. It is a subset because in practice, the system administrator may limit configurations by setting constraints on which software components must or cannot be co-located. The iMobile-EE architecture allows new components to be brought up and old ones shut down while the system is running. Thus, no code modifications were required for implementing these operating submodes.

## 3.3 Adaptation Model Formulation

Based on the preceding discussion of the operational submodes, the set of operating modes for the messaging system is defined as $M = C \times \{M_{imm}, M_{delayed}, M_{noack}, M_{transfer}\}$. Each mode specifies both the acknowledgment-processing submode and the component configuration. The adaptation decisions to transition between these modes depend on the workload of the system as defined by the rate of incoming messages $\lambda \in \mathbb{R}$, the set of hosts that have not failed (a member of $2^H$), and the number of acknowledgments stored at the gateways (that need to be transferred to the back-end servers later), which is a member of the set of natural numbers $\mathbb{N}$. Hence, the world state-space is defined as $W = \mathbb{R} \times 2^H \times \mathbb{N}$. There are three types of external events: host failures that change the set of operational hosts, changes in workload rate, and changes in the number of delayed acknowledgments at the gateways. For simplicity, it is assumed that hosts fail independently of each other with negative exponential time between failures.[3] We do not explicitly include repair events as part of the predictive model. Therefore, the models optimize the worst-case behavior and always contain a system-failure absorbing state. However, that does not mean that if the system is repaired, the models become useless: it only means that the model does not condition the adaptations it proposes on the fact that repairs will be performed.

The world state $W$ formulated above has a huge state space and hence is not amenable to numerical solution in its unmodified form. This problem is tackled using two approximation techniques: aggregation and decomposition. We employ aggregation by a) setting upper and lower bounds on the workload rate and the number of outstanding acknowledgments using the system specification and the maximum buffer space available on the gateways, respectively, and b) by partitioning the finite range of the workload rates and the number of acknowledgments into a small number of bins. Each bin is associated with a single aggregate parameter value that is computed as the mean of the bin's range. All parameter values are mapped to the bin whose range they fall in, and are represented by the aggregate value associated with the bin. We are studying how the binsize affects the policy generated by the model, and the rewards obtained by the system.

We employ decomposition by using the observation that the time-scales at which workload changes and failures occur are different. Failures usually occur far less frequently than workload changes (especially when hosts fail independently of each other). Therefore, the possibility of failures beyond the first one is unlikely to have any substantial effect on adaptation actions taken in response to a changing workload. Moreover, we assume that component reconfiguration is done only in response to failure. Using these observations, we are able to decompose the adaptation model into two smaller models: a) a failure adaptation model that contains only failure events and component configurations $C$, and b) a workload adaptation model for each component configuration that considers only the acknowledgment-processing modes and the workload rate and number of outstanding acknowledgments (both in the form of aggregated bins). The workload adaptation model depends on the current software configuration, and is recomputed with the new configuration whenever there is a change in the configuration as a result of a failure.

The final element of the adaptation model is the reward specification, and that is based on the timeliness goals of

[3]This is typically true for hardware failures, but not for software failures or malicious attacks. Such failure models can be represented using the adaptation model, but are left as future work.

the system. The timeliness guarantees are specified by a target value for the average delivery time for a message. If the system manages to maintain its average response time below a specified target value, then it receives a full reward for each message delivered and acknowledged, but only a partial reward for each message whose acknowledgment was lost or refused. If the system's delivery time increases beyond the target, it is penalized proportional to the difference between the target and actual values. The average response times for each system configuration and acknowledgment processing mode are computed as a function of the workload using a product-form queuing network model [3], and updated online using data collected as the system executes. Then, the per-message reward is computed using the average response time, target average response time, and the acknowledgment-processing mode. Finally, the per-message reward is multiplied by the workload to obtain the reward for that world-state and operating mode combination.

### 3.4 The Adaptation Controller

As mentioned in Section 2, the adaptation controller is a software component introduced into the system to generate the adaptation policies. In the enterprise messaging system, the controller generates both actions that change the acknowledgment processing mode, and reconfiguration actions. The failure model and its policy are precomputed using assumed values for failure rate, average workload, and co-location policy (it could be recomputed if any of those parameters change substantially). The policy is then propagated by the adaptation controller to all hosts in the system, and stored locally on them. This propagation has to be atomic so that all the hosts have the same policy at all times. Whenever a failure occurs and is detected, each host locally looks up the new configuration specified by the policy, and starts up or stops the required components.

The optimal policy for performance adaptation is computed at the beginning of system execution, and subsequently thereafter whenever there is a failure that causes a change in the system configuration. While the policy is being computed by the controller, the previous acknowledgment processing mode is used. In addition to the system configuration, the controller needs several parameters to compute the rewards for each state in order to compute the adaptation policy. They include the average message and acknowledgment processing time both at the gateways and the back-end servers, the average number of recipients for a message, and the probabilities that the recipients will use each gateway type. These parameters are obtained through system instrumentation.

In order to track the state for the performance model, the HTTP gateways keep track of the rate of the incoming message stream. Similarly, each email and voice gateway measures the number of outstanding acknowledgments. These measurements are periodically propagated to the controller, which computes the aggregate values and places them in the corresponding bins. If any of the bins have changed, then the controller looks up the acknowledgment processing mode corresponding to the new state, and propagates it to the gateways. Since having different acknowledgment processing modes at different gateways (for a short period of time) does not hinder the safety or correctness of the system, the decisions are propagated in an uncoordinated manner.

## 4. FUTURE WORK AND CONCLUSIONS

In this paper, we have presented an approach based on Markov decision theory that constructs controllers for self-managing systems that are model-based (thus predictive), and can handle delayed and long-term effects. We believe that the approach is useful for optimization of combined performance and dependability metrics. The application of the approach to an enterprise messaging system was described. State-space explosion is a challenge with this approach, and we briefly mentioned some ways in which we are tackling the problem in the context of the application. We have implemented the model generation and solution components that solve the models. Currently, we are in the process of implementing and integrating the models and the controller for the enterprise messaging system application and experimenting with model solution times and accuracy. In future work, we will evaluate the benefits of these models using both experimental evaluation on the system (for the workload adaptation model) and simulation (for the failure adaptation model).

## 5. REFERENCES


[1] M. Abdeen and M. Woodside. Seeking optimal policies for adaptive distributed computer systems with multiple controls. In *Third Intl. Conf. on Parallel and Dist. Computing, Applications and Technologies*, Kanazawa, Japan, Sept. 2002.

[2] T. Abdelzaher, K. G. Shin, and N. Bhatti. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Trans. on Parallel and Dist. Sys.*, 13(1):80–96, 2002.

[3] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *J. ACM*, 22(2):248–260, 1975.

[4] W.-K. Chen, M. Hiltunen, and R. Schlichting. Constructing adaptive software in distributed systems. In *Proceedings of the 21st Intl. Conf. on Dist. Computing Sys.*, pages 635–643, Mesa, AZ, Apr 2001.

[5] Y.-F. Chen, H. Huang, R. Jana, T. Jim, M. Hiltunen, S. John, S. Jora, R. Muthumanickam, and B. Wei. iMobile EE: An enterprise mobile service platform. *Wireless Networks*, 9(4):283–297, 2003.

[6] H. de Meer and K. S. Trivedi. Guarded repair of dependable sys. *Theoretical Computer Sci.*, 128:179–210, 1994.

[7] L. J. Franken and B. R. Haverkort. Reconfiguring distributed systems using Markov-decision models. In *Trends in Dist. Sys.*, Aachen, Oct. 1996.

[8] D. Henriksson, Y. Lu, and T. Abdelzaher. Improved prediction for web server delay control. In *Proc. of 16th Euromicro Conf. on Real-Time Sys.*, pages 61–68, Catania, Sicily, 2004.

[9] M. Hiltunen and R. Schlichting. Adaptive distributed and fault-tolerant systems. *Computer Sys. Sci. and Eng.*, 11(5):125–133, Sep 1996.

[10] S. Parekh, N. Gandhi, J. L. Hellerstein, D. M. Tilbury, T. S. Jayram, and J. Bigus. Using control theory to achieve service level objectives in performance management. *Real-Time Sys.*, 23(1-2):127–141, 2002.

[11] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Intersci., 1994.

[12] K. G. Shin, C. M. Krishna, and Y.-H. Lee. Optimal dynamic control of resources in a distributed system. *IEEE Trans. on Software Eng.*, 15(10):1188–1198, Oct. 1989.

[13] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[14] J. Zinky, D. Bakken, and R. Schantz. Architectural support for quality of service for CORBA objects. *Theory and Practice of Object Sys.*, 3(1), 1997.