# A User-Centric Framework for Improving a Distributed Software System's Deployment Architecture

Sam Malek

Computer Science Department
University of Southern California
Los Angeles, CA 90089-0781 U.S.A.
malek@usc.edu

## Abstract

*A distributed system's deployment architecture can have a significant impact on its QoS. Furthermore, the deployment architecture will influence users' satisfaction, as users typically have varying QoS preferences for the system services they access. Finding a deployment architecture that will maximize the users' overall satisfaction is a challenging, multi-faceted problem. We propose to develop a framework that can be tailored and instantiated to address this problem in its many variations. The framework is accompanied with tool support, which allows it to be used in practice and evaluated in many representative scenarios.*

## 1. Research Problem and Importance

A growing class of mobile and distributed systems are frequently challenged by the fluctuations in the system parameters: network connectivity, bandwidth, reliability of hosts, availability of battery power, etc. Furthermore, the different users' usage of the functionality (i.e., *services*) provided by the system and the users' quality of service (QoS) preferences for those services will differ, and may change over time. We define a service as a separately identifiable functionality provided by a system that is accessed via a specific access point (i.e., a human-user or software interface) and realized via a collaboration of a subset of the system's components.

For any such large, distributed system, many *deployment architectures* (i.e., mappings of software components onto hardware hosts) will be typically possible. Some of those deployment architectures will be more effective than others in delivering the desired level of service quality to the user. For example, a service's latency can be improved if the system is deployed such that the most frequent and voluminous interactions among the components involved in delivering the service occur either locally or over reliable and capacious network links. However, the problem of finding the optimal (or even reasonable) physical locations for software components becomes quickly intractable for a human engineer if multiple QoS dimensions (e.g., latency, security, availability, power usage) and multiple users' preferences must be considered simultaneously, while taking into account any additional constraints (e.g., component X may not be deployed on hosts Y and Z). Figure 1 shows an overview of the problem, its inherent complexity, and the relationships among its elements.

In this research, we consider the problem of finding a deployment architecture such that the QoS preferences accrued by a collection of distributed end-users are addressed, i.e., that the *utility* of the system to all of its users is maximized. We would like our solution to be applicable to a wide range of *application scenarios* (i.e., differing numbers of users, hardware hosts, software components, application services, QoS dimensions, etc.). However, a widely applicable solution to this problem is challenged by the following: (1) A very large number of system parameters influence *QoS dimensions* (e.g., security, availability) of a software system; while it may be possible to identify a subset of *system parameters* (e.g., network bandwidth, frequencies of interactions) that influence the majority of QoS dimensions, it may not be possible to identify all of them. (2) Many services and their corresponding QoS influence the users' satisfaction. (3) Different QoS dimensions may be conflicting (i.e., improving one may degrade another), and users with different priorities may have conflicting QoS preferences. Fine-grain trade-off analysis without relying on simplifying assumptions (e.g., a particular definition of a QoS objective, predetermined constraints) is challenging. (4) Different application scenarios require different algorithmic approaches. For example, system's size, users' usage of the system, stability of the system's parameters, and its degree of (de)centralization will likely determine the best algorithm for finding the best deployment. (5) Traditional software engineering tools are not applicable to this problem. Therefore, engineers have to spend a significant amount of time adapting tools intended for different purposes to the problem of improving system deployment. In turn, this limits the potential for reuse and cross-evaluation of the solutions.

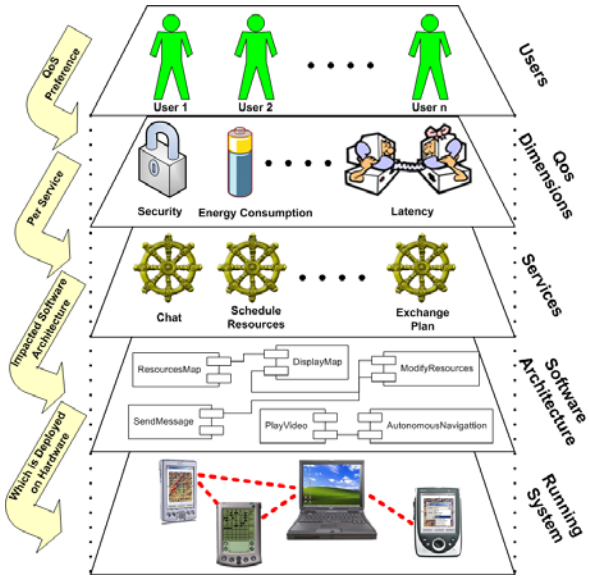


**Figure 1. Problem Overview.**

## 2. Prior Research

Several researchers have considered modifying a software system's deployment architecture to improve a specific QoS dimension of the system. I5 [1], proposes the use of the binary integer programming model for generating an optimal deployment of a software application over a given network that minimizes the overall remote communication. Coign [2] provides a framework for distributed partitioning of COM applications across the network. Kichkaylo et al. [3], provide a model, called Component Placement Problem (CPP), for describing a distributed system in terms of the constraints on its deployment, and an AI planning algorithm, called Sekitei, for solving the CPP model. However, while all of the above works recognize that the problem of distributing an application is NP hard, none provide approximative solutions for such cases.

In our prior work [5,7], we devised a set of algorithms for improving system's availability by finding an improved deployment architecture. The novelty of our approach was a set of approximative algorithms that scaled well to large distributed software systems with many components and hosts. However, our approach was limited to a predetermined set of system parameters, and a predetermined definition of availability.

None of the above approaches (including our previous work) considers the system users and their QoS preferences. Furthermore, none of these approaches attempt to improve more than one QoS dimension of interest. Finally, no previous work has considered users' QoS preferences at the granularity of the application-level services. Instead, the entire distributed software system is treated as one service with one user, and a particular QoS dimension serves as the only QoS objective.

## 3. Research Hypotheses

Our research is based on the following hypotheses.

**Optimization Algorithms.** Finding the optimal deployment architecture is an exponentially complex problem: there are $h^c$ possible deployment architectures, where $h$ is the number of hosts, and $c$ is the number of components. Therefore, it may be impossible to invest the necessary time to find the optimal solution. We hypothesize that *an algorithm of at most polynomial complexity in the number of components and hosts, and linear in the numbers of QoS dimensions, users, and services can be devised with the ability to find a deployment architecture such that (1) the overall utility to system users will be very close to a target (e.g., known optimal) architecture's utility, or (2) when a known optimal architecture does not exist, the overall utility will improve significantly more than the statistical average utility, which is the average utility of a set of randomly selected deployment architectures.*

**Sensitivity to Users and Services.** Some users of the system may be more important than others. Similarly, some services may be more critical than others. We hypothesize that *an optimization algorithm can be devised that performs fine-grain trade-off analysis, such that given two identical application scenarios X and Y: (1) if they only differ in the priority of a user U such that he has a higher priority in X than in Y, then after executing the algorithms on both X and Y, the overall utility gain for U in X is greater than or equal to his utility gain in Y; or* (2) *if they only differ in the criticality of service S such that it is more critical in X than in Y, then after executing the algorithm on both X and Y, the overall QoS improvement for S in X is greater than or equal to its improvement in Y.*

**Decentralization.** Centralized algorithms depend on the existence of a host with the global knowledge of the system. However, this is not feasible in a growing class of decentralized systems, where each host has only partial knowledge of the system. We hypothesize that *a decentralized optimization algorithm can be devised such that (1) when there is a modest lack of knowledge (defined as the situation where each host on average does not know about 20% or less of the hosts in the system) the algorithm finds solutions that on average come very close to the best solution produced by the centralized optimization algorithms, and (2) while there are no completely disconnected hosts, the solution accuracy degrades gracefully as the lack of knowledge increases on each host, such that the decrease rate in the solution accuracy is significantly lower than the increase rate in the lack of knowledge.*

**Algorithmic Trade-Offs.** There exist inherent trade-offs among the deployment improvement algorithms. Each algorithm has its own unique properties that make it more suitable to a class of systems. We hypothesize that *it is possible to determine the best algorithm for execution in terms of the accuracy of the solution and the performance of the algorithm given the system's architectural style (e.g., client-server vs. peer-to-peer), its stability (amount of fluctuation in system parameters, which impacts the available time for estimation), centralization, the number of system parameter constraints (highly vs. lightly constrained), and the complexity of the application scenario (which includes number of hosts, components, logical links, physical links, users, services, and QoS dimensions).* We also hypothesize that *an autonomic solution can be devised that given the monitored characteristics of an application scenario determines the best deployment improvement algorithm for execution at runtime.*

## 4. Approach

In support of evaluating the hypotheses, we have developed a framework for improving a distributed system's deployment that relies on the notion of *QoS utility function*, which indicates a user's (desired) degree of satisfaction with improvements in a given QoS dimension. We leverage each user's utility functions and employ a common strategy of transforming the multidimensional objective (i.e., the vector of QoS dimensions) to a single scalar value. This transformation allows us to resolve trade-offs inherent in our multi-dimensional optimization problem. The framework's ideal objective is to maximize the overall utility, i.e., the cumulative satisfaction with the system by all its users. Given an application scenario, the engineer instantiates (configures) the framework by defining the appropriate system parameters and the QoS of interest. The framework is then populated with the actual data from a distributed application and users' preferences for the QoS dimensions of each application service. Each user's preferences are adjusted based on his priorities and importance. Afterwards, one of the algorithms supplied by the framework is used to find an improved deployment. Finally, the solution is effected by (re)deploying the system.

We demonstrate some aspects of our approach on a simple application scenario that consists of two hosts, two components, two QoS dimensions, one user, and one service. In this scenario, there are four possible deployment architectures. Figure 2a shows the quantification of each deployment in terms of the two QoS

dimensions of latency and durability. Each QoS dimension is quantified based on the various system properties. For example, latency of a service can be quantified as the product of the number of messages exchanged between software components and the network transmission delays. Similarly, durability of the service can be quantified as the ratio of available battery power on each device to the average energy consumption of software component on each device. Due to space constraints we cannot present our complete analytical model for quantifying latency and durability, which can be found in [6]. However, note that the framework's algorithms are independent of the analytical model used for quantifying the QoS dimensions. As will be discussed below, this is because the framework relies on the rate of change in QoS dimensions, as opposed to actual quantitative values produced by the analytical model.

In this scenario, since the objective is to maximize durability and minimize latency, with the exception of deployment 4, which has both a higher latency and a lower durability than deployment 3, all the other three deployments present some kind of a trade-off. Therefore, it is not possible to determine the optimal deployment. This is a frequently encountered phenomenon in multi-dimensional optimization problems, and is known as Pareto Optimal. As mentioned earlier we can solve this problem by considering users' preferences. Figure 2b shows the user's utility functions for the two QoS dimensions. For example, it shows that for 25% increases in latency and durability, the user has specified utilities of -1 and 2, respectively. The shown utility functions are linear, but the framework places no restrictions on the type of functions that represent users' preferences. In fact, since typically users may not be able to express their preferences in terms of complex mathematical functions, we first need to elicit and express users' preferences in terms of a set of discrete data points (e.g., 50% decrease in latency has a utility of 2 and so on), and then use one of the numerous curve fitting techniques (e.g., Regression, Interpolation) to determine a function that approximates the data points most accurately.

To determine the utility of changing the initial deployment, we first determine the rate of change for each QoS dimension (i.e., amount of change in a QoS dimension if we were to modify the current deployment) from Figure 2a, then look up the utility associated with each rate of change from Figure 2b, and finally aggregate the utilities. Optimal deployment for the system is the one that has the highest total utility. Figure 2c shows the total utility of changing the system's deployment in our example based on the assumption that deployment 2 is the initial deployment of the system. As shown in Figure 2c, deployment 3 achieves a total utility of 2, which is the optimal deployment for this system.

While an engineer can apply this approach to larger problems as well, doing this manually becomes infeasible very fast. In fact, since the number of possible deployments grows exponentially in the number of hosts and components, and the number of utility functions that would need to be considered grows polynomially in the number of users, services, and QoS dimensions, finding the optimal solution algorithmically also becomes infeasible very fast. As will be discussed in the next section, a significant contribution of this research is the development of the appropriate algorithms and tool support to assist the engineers in exploring and improving system's deployment architecture.

## 5. Preliminary Work

The theoretical underpinnings of the work, which includes an extensible formal model of a distributed system's deployment architecture and the accompanying generic algorithms, are independent of any implementation platform. Our preliminary work [5,6] on the theoretical aspects of the framework has resulted in the development of several algorithmic solutions to this problem: *Mixed Linear and Non-Linear Integer Programming*, *Genetic*, *Greedy*, and *Market-based*. Our initial results indicate that there are significant trade-offs between these algorithmic solutions, which make each algorithm suitable for a particular application scenario. For example *Mixed Linear Integer Programming* is the only algorithm that finds the optimal solution. However, since it is an exponentially complex algorithm, it is applicable to systems that are either relatively small, stable, or have highly constrained architectural styles that impose strict locational constraints on the deployment of components (e.g., *Client-Server*). On the other hand, optimization algorithms such as *Greedy* and *Genetic* are suitable for systems that are large, unstable, or have flexible architectural styles that do not impose any locational constraints (e.g., *Peer-to-Peer*).

We are in the process of profiling each algorithm and determining its unique characteristics. This will help us to develop heuristics that could aid the engineer in the selection of the best algorithm for execution. We also plan to leverage the above results to devise an autonomous agent that selects the best algorithm for execution and effects the improved deployment architecture by redeploying (part of) the system at runtime.

The theoretical results discussed above will be realized on top of an integrated tool suite, which allows the engineer to create a deployment model, instantiate the model for an application scenario, and use one of the provided algorithms for improving its architecture. For constructing the tool suite we will leverage our previous work on a customizable deployment analysis environment (DeSi [8]) and an extensible architectural middleware (Prism-MW [4]). Prism-MW provides the ability to implement, (re)deploy, execute, and monitor a distributed system in terms of its architectural components. DeSi provides the ability to model

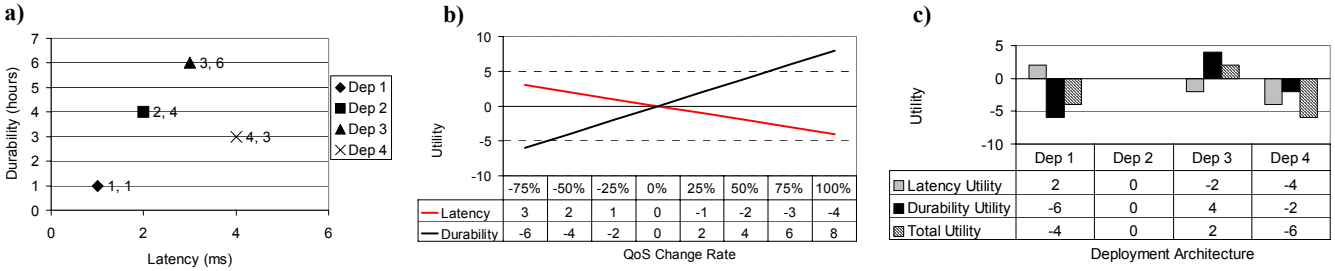


**Figure 2. Simple example: a) quality of four possible deployments, b) utility functions, and c) utility achieved assuming system's initial deployment is deployment 2.**

the system's deployment architecture, visualize and assess its architecture, and improve it via one of the deployment improving algorithms (shown in Figure 3a).

We will enhance DeSi to model multiple users, arbitrary QoS dimensions, user preferences, and services provisioned by the system. We will also incorporate our algorithms into DeSi. Furthermore, we will integrate DeSi with Prism-MW, such that: (1) DeSi's deployment models are populated by the runtime monitoring data from Prism-MW, and (2) once an improved deployment is selected for effecting, DeSi can send the appropriate (re)deployment commands to Prism-MW.

A prototype of this functionality has already been constructed. As an illustration, Figure 3b depicts a distributed system of 5 hosts and 35 components that is monitored and deployed on top of Prism-MW. The *Deployer* and *Admin* components in Figure 3b correspond to meta-level components provided by Prism-MW for coordinating the monitoring and (re)deployment of software components. DeSi's *Monitor* and *Effector* components (shown in Figure 3b) provide the interface between Prism-MW and DeSi, which is wrapped via a *Prism-MW Adapter*. Once the *Deployer* component determines that the monitoring data is stable, it sends the data to DeSi, which populates its model. One of the algorithms is then selected and executed for improving the system's deployment architecture. The results is reported back to the *Deployer*, which coordinates the redeployment of the system with the help of the *Admin* components.

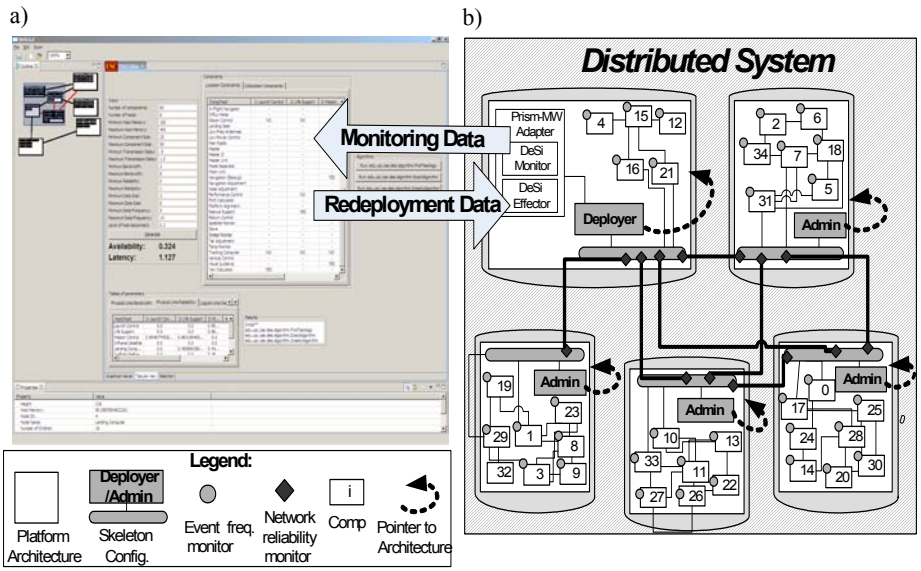


**Figure 3. Illustration of the approach: a) DeSi's tabular view of system's deployment data, b) a system running on top of Prism-MW that is monitored and managed by meta-level components.**

## 6. Evaluation

The approach will be evaluate on a large number of real and simulated distributed systems. In fact, it is currently being applied and evaluated on MIDAS, which is a distributed sensor network application developed on top of Prism-MW as part of a joint research project with Bosch Research and Technology Center. The work will be evaluated on application scenarios with: 1) multiple QoS dimensions (e.g., availability, latency, communication security, and durability), 2) multiple real and simulated users with varying QoS preferences, and 3) systems with different characteristics (small vs. large, stable vs. unstable, centralized vs. decentralized). The algorithms will be evaluated on their ability to improve user's QoS preferences. The tool suite will be evaluated on its ability to promote reusability and cross-evaluation.

## 7. Research Contribution

As mentioned earlier, the related approaches have relied either on simplifying assumptions (e.g., single QoS dimension) or characteristics of specific application scenarios (e.g., a prespecified model of the system), which has restricted their applicability. Unlike previous works, we have addressed this problem as a multi-dimensional optimization problem, and by leveraging users' preferences, we have been able to resolve inherent trade-offs in conflicting QoS dimensions. We have provided an extensible system modeling approach that can be leveraged across different application scenarios and a suite of generic multidimensional optimization algorithms that can be leveraged to improve arbitrary QoS dimensions. Furthermore, we have developed a customizable tool suite that can be leveraged for visually assessing hypothetical or real systems, and performing actual monitoring and (re)deployment of software components.

## 8. References


[1] M. C. Bastarrica, et. al. A Binary Integer Programming Model for Optimal Object Distribution. *Int'l. Conf. on Principles of Distributed Systems*, France, Dec. 1998.

[2] G. Hunt, et. al. The Coign Automatic Distributed Partitioning System. *Symposium on Operating System Design and Implementation*, New Orleans, Feb. 1999.

[3] T. Kichkaylo et. al. Constrained Component Deployment in Wide-Area Networks Using AI Planning Techniques. *Int'l. Parallel and Distributed Processing Symposium*. April 2003.

[4] S. Malek, et. al. A Style-Aware Architectural Middleware for Resource-Constrained, Distributed Systems. *IEEE Trans. on Software Engineering*, March 2005.

[5] S. Malek, et. al. A Decentralized Redeployment Algorithm for Improving the Availability of Distributed Systems. *Int'l Conf. on Component Deployment*, Grenoble, France, Nov. 2005.

[6] S. Malek, et. al. A User-Centric Approach for Improving a Distributed Software System's Deployment Architecture. Tech. Report USC-CSE-2006-602, 2006.

[7] M. Mikic-Rakic, et. al. Improving Availability in Large, Distributed, Component-Based Systems via Redeployment. *Int'l. Conf. on Component Deployment*, Grenoble, France, 2005

[8] M. Mikic-Rakic, et. al. A Tailorable Environment for Assessing the Quality of Deployment Architectures in Highly Distributed Settings. *Int'l. Conf. on Component Deployment*, Edinburgh, UK, May 2004.