

Research Statement

My research interests are in the field of software engineering. A common theme, and long-term goal, of my research is *development and evolution of reliable, highly adaptable, distributed, large-scale software systems*. Practitioners have traditionally faced many problems with achieving support for software reuse, reconfiguration, and extension in a manner that preserves desired system properties and relationships. These problems are becoming even more important with the recent emergence of small, resource-constrained, and highly-mobile computing platforms.

The main hypothesis of my research is that an explicit architectural focus can remedy many of these difficulties. I therefore focus on *software architectures* as a key to developing mechanisms for engineering reliable, flexible, highly-distributed, large-scale software. Architectures present a high-level view of a system, enabling developers to abstract away the unnecessary details and focus on the “big picture.”

My work to date has investigated a wide range of issues in the area of software architectures: software (re)deployment, architectural middleware, architecture-based software evolution, self-reconfigurable and self-healing systems, and architecture-level configuration management. Below I discuss each one of these areas in more detail. I then conclude with an overview of my planned future research directions.

Software (re)deployment. My dissertation research focuses on techniques for supporting disconnected operation of distributed systems, i.e., improving the availability of distributed systems in the face of (temporary) network failures. I have extensively studied existing disconnected operation techniques in the areas of distributed file systems, database systems, code mobility, and ad-hoc networks. This study has resulted in a preliminary taxonomy of the existing techniques and identification of the areas they do not adequately address. A key observation about distributed systems is that the assignment of software components onto hardware nodes (i.e., a system’s deployment architecture) greatly influences the system’s availability in the face of connectivity losses. However, determining a software system’s deployment that will maximize its availability is an exponentially complex (NP-hard) problem that depends on a number of system parameters: frequencies of interaction among software components, volumes of data exchanged among them, the components’ memory requirements, reliabilities and bandwidths of connections among hardware hosts, available memory on each host, and so on. I have developed a suite of polynomial-time approximation algorithms that improve system-wide availability while taking into account the above system parameters. These algorithms have experimentally shown very good performance. I have also implemented an environment that allows observation of the effects of modifying system parameters (e.g., network connectivity, interaction frequency) at runtime and comparison of the behavior, performance, and precision of different algorithms. In order to automate the simulation of different deployment architectures, the environment supports automatic generation of canonical components with desired processing delays, network links with desired reliability and bandwidth, hosts with desired memory and CPU characteristics, and so on. This is a very rich research area and my dissertation has barely “scratched its surface”. I have recently begun expanding this work to include decentralized systems, and to render my approach more broadly applicable by including additional system properties, such as latency and reliability, possibly in tandem.

Architectural middleware. To be truly useful in a development setting, software architectures must be accompanied by support for their implementation. This is particularly important in the context of highly distributed, decentralized, mobile, and long-lived systems, where there is an increasing risk of “architectural drift” unless there exists a clear relationship between the architecture and its implementation. As part of my dissertation research I have worked on the design and implementation of Prism-MW, a highly extensible, efficient, and scalable middleware platform that supports implementation of distributed, mobile, and resource-constrained systems in a number of architectural styles (e.g., pipe-and-filter, client-server, publish-subscribe, peer-to-peer). I have leveraged Prism-MW’s extensibility to develop implementation-level support for the problem of disconnected operation discussed above, including component deployment, system monitoring, component mobility, and runtime system reconfigurability. Prism-MW has been used in several courses at USC, and successfully evaluated by three industrial organizations. One of the organizations, NASA’s Jet Propulsion Laboratory, recently reimplemented their data-grid platform called OODT on top of Prism-MW, enabling it to significantly improve OODT’s adaptability, meet its performance targets, and decrease the resulting grid application size by two orders of magnitude.

Architecture-based software evolution. I have investigated a specific instance of software evolution which involves component upgrade, i.e., incorporating into a system a new version of an already existing component. Prior to performing component upgrade it is important to assess the behavior of a new component version, including whether the new version correctly preserves the functionality carried over from the old version, whether the new version introduces new errors, whether there is any performance discrepancy between the old and new versions, and so forth. Specifically, I have designed and developed special-purpose software connectors, called multi-versioning connectors (M-V-Cs), to ensure dependable system composition and evolution. M-V-Cs allow multiple versions of a component to execute “side-by-side” in the deployed system. M-V-Cs unintrusively collect and analyze the execution statistics of multiple component versions and perform comparisons of their performance (i.e., execution speed), reliability (i.e., number of failures), and correctness (i.e., ability to produce expected results). Although this thread of research has resulted in several publications, it has not been the focus of my dissertation, and a number of interesting issues still remain to be explored.

Architecture-level configuration management. As part of a collaborative effort related to my work on M-V-C, I have participated in the design and implementation of a software architecture evolution environment, called Mae. Mae facilitates an incremental design process in which all changes to architectural elements are integrally captured and related. This project has resulted in a rich system model that combines architectural concepts with those from the field of configuration management. This system model has been shown to enable precise capture of system characteristics needed to support architectural evolution. While this is not a part of my immediate research goals, I intend to integrate Mae with my disconnected operation support to enable integrated architecture-level system modeling, analysis, implementation, deployment, and evolution.

Self-healing systems. In order to improve the understanding of the emerging class of large, complex, mobile, distributed, but frequently resource-constrained systems, I have also worked on identifying the characteristics of architectural styles suitable for self-healing systems. Self-healing systems exhibit the ability to adapt themselves at runtime to handle situations such as resource variability, changing user needs, and system faults. As part of this study, I have identified a set of

requirements that an effective style for self-healing systems should satisfy, and have classified these requirements along five dimensions I have identified as distinguishing characteristics of architectural styles: external structure, topology rules, behavior, interaction, and data flow. One aspect of self-healing I have specifically focused on in my thesis research is support for disconnected operation. However, many other aspects of self-healing remain open research questions.

Future research directions. In addition to the issues brought up in the above discussion, as part of my longer-term research goals I plan to investigate several other questions, including

- How can we address disconnected operation in a decentralized setting?
- How can we improve other non-functional properties of distributed systems (e.g., efficiency, security, latency, survivability) by employing software redeployment? Which parameters influence these non-functional properties? How can we extract these parameters from a given system? How can we develop efficient techniques for achieving multiple non-functional system characteristics simultaneously?
- What are effective middleware solutions for ensuring different non-functional properties of running systems? How can these solutions be adapted to perform well in highly decentralized, partially disconnected systems?
- How can we effectively leverage other disconnected operation techniques (e.g., replication, caching, queueing of remote invocations) to improve availability? How do we quantify the effects of these techniques on a system's non-functional properties? How do we minimize the negative impact of replication and caching while maximizing the benefit of near-continuous availability?
- How can I leverage M-V-Cs in supporting disconnected operation? What is the relationship between multi-versioning and replication? What are the implications of using multi-versioned components on system correctness and efficiency?
- What are the issues in supporting system awareness, location transparency, and real-time requirements in highly-distributed, mobile, possibly embedded systems?
- How do we effectively support different aspects of self healing (e.g., robustness, autonomy) in an architectural style and its supporting middleware? What is the appropriate response of self-healing systems to network disconnections?