

## Research Statement

My research lies in the field of software engineering. A common theme, and long-term goal, of my research is *development and evolution of adaptable and dependable large-scale software systems*. Software practitioners have traditionally faced many problems with designing, implementing, deploying, and evolving software modules and/or systems. These problems are often the result of poor understanding of a system's overall architecture, unintended dependencies among its modules, decisions that are made too early in the development process, over-reliance on specific implementation technologies, and so on. Traditional techniques that are intended to remedy these problems (e.g., separation of concerns or isolation of change) are only partially adequate in the case of development with pre-existing, large, multi-lingual components that originate from multiple sources.

The main hypothesis of my research is that an explicit architectural focus can remedy many of these difficulties and enable flexible construction and evolution of large software systems. I have therefore centered on *software architecture* as a key to developing techniques, tools, and methodologies for engineering flexible, large-scale software. Architecture presents a set of high-level design views of a system, enabling developers to abstract away the unnecessary details and focus on the system's building blocks (*components*), interactions (*connectors*), their structure (*configuration*), and key *properties*. A given system's architecture frequently adheres to one or more *architectural styles*, which are design guidelines shown in practice to result in systems with certain desired characteristics.

My Ph.D. thesis was indeed in the area of software architecture. It introduced and investigated C2, an architectural style for graphical user interface (GUI) intensive systems [J12].<sup>1</sup> Specifically, my thesis focused on software architecture modeling and analysis mechanisms within the C2 style to support design-time system evolution. This work has helped to motivate and inform my research over the past five years. At the same time, the scope of my research during this period has broadened significantly. I have conducted work in four related areas:

1. the principles underlying software architectures;
2. software architecture modeling and analysis;
3. the relationship of software architectures to other software system artifacts; and
4. the role of software architecture at system runtime.

In doing so, I have expanded my focus beyond a single style (C2) to a large class of architectural styles for modern distributed systems (client-server, pipe-and-filter, peer-to-peer, publish-subscribe, and so on). Another significant facet of my work has been applying the principles of software architecture to the emerging class of mobile, possibly embedded systems running on resource-constrained platforms.

In the remainder of this document, I will outline the challenges and major results of my work in the above four areas. I will also highlight the research grants that have supported the different efforts, and technology transition opportunities that have emerged from my work. I will conclude the research summary by discussing my future research plans. While I use first person singular in my exposition for convenience, a number of the described research results emerged from collaborative efforts, as indicated in the references cited.

---

1. Full citations for the referenced publications can be found in the *Curriculum Vitae*, at <http://sunset.usc.edu/~nenocv/CV.pdf>

## 1. Underlying Principles of Software Architecture

Software architecture has emerged as a research discipline within the past 15 years. A number of key concepts, characteristics, and relationships within this area still remain unexplored and poorly understood. A part of my work has tried to remedy this shortcoming. My work in this area comprises four broad-based, foundational studies. These studies have served to inform and directly motivate my other research efforts.

**Architectural Description** — The first foundational study deals with the nature of software architecture modeling notations, referred to as architecture description languages (ADLs). I have developed a classification framework for ADLs that identifies the key dimensions of architectural description at the level of individual components and connectors, as well as entire architectures. I have applied this framework to a large cross-section of existing ADLs. In the process, I have also shown how the framework can be used for ADL comparison. The paper that resulted from this work [J9] is included in this packet. This paper has become widely cited and has been selected by Grady Booch, a software design pioneer, as one of the 50 “seminal papers in software architecture”. I have recently also conducted a study of the applicability of existing ADLs to the domain of embedded systems [C10], and a collaborative project whose goal was to assess the use of multiple ADLs in concert in an actual embedded system project [C2, C3].

**Architectural vs. Design Models** — A closely related study to that of ADLs has dealt with the relationship of architectural modeling, which has only relatively recently become a topic of interest to software engineering researchers, with design modeling, which has been studied comparatively much longer. Specifically, I have selected the Unified Modeling Language (UML) as the target design language for this study. UML is an extensible collection of loosely integrated notations that are intended to model different aspects of a software system’s design. UML has become a de-facto standard software design language. Soon after its emergence, it was frequently claimed that UML can also be used for architectural modeling. My study was the first to show the precise relationship between ADLs and UML, and to provide evidence of UML’s strengths, but also deficiencies as a possible ADL. This study resulted in a series of publications [J7, C47, C50, I6].

**Software Connectors** — One distinguishing aspect of software architectures is their explicit focus on and support for software connectors. Connectors capture interactions among components and are widely considered to be first-class modeling constructs in software architectures. My third foundational study has dealt with understanding the nature, types, and key characteristics of software interactions. The study resulted in the identification of four fundamental roles of software connectors (communication, coordination, conversion, and facilitation), eight connector types, and a comprehensive classification of their dimensions and values [C18, C37, C41, I5]. The primary publication that emerged from this study [C37] has been widely cited as the definitive study of its kind to date and is included in this packet.

**Architectural Styles** — My fourth foundational study has investigated the shared properties of architectural styles. The characterization of software connectors directly enabled this study. While components are application-specific processing and storage elements in a software architecture, connectors are application-independent (and thus potentially style-wide) interaction facilities. My analysis of a large number of distributed systems’ styles has identified five orthogonal dimensions of style characterization: data, structure, topology, behavior, and interaction [C19]. This characterization has been used as a basis for supporting style-based system design and development in the emerging world of handheld computing (further discussed below) [J5]. This novel character-

ization also has allowed the identification of a small number of recurring architectural *primitives* that can be composed to describe the five style dimensions [C6, C12]. The resulting project will be referred to as *Alfa* below.

**Research Grants** — Two primary sources of support for these studies have been my NSF CAREER [G4] and DARPA DASADA [G7] grants. The study of ADLs in the context of embedded systems was supported by the NASA HDCP [G3] grant. A recent grant from Boeing [G5] is geared to applying Alfa in the development of architectural styles for military tactical radio systems.

**Ph.D. Students** — The studies of software connectors and architectural styles (including the Alfa project) have been spearheaded by my doctoral student Nikunj Mehta, who graduated in September 2004. The study of the applicability of ADLs to the domain of embedded systems has been led by my doctoral student Roshanak Roshandel.

## 2. Software Architecture Modeling and Analysis

My study of ADLs mentioned above has indicated a number of open issues in architectural modeling and analysis. I have tried to address those in my work. Specifically, I have tried to enrich architectural models and analysis techniques to include the structural (external interfaces), functional (static and dynamic behaviors), interaction (protocols), and non-functional (e.g., availability, reliability, latency) system properties. This work has also focused on architectures for families of applications.

**Architectural Modeling** — I have leveraged my study of ADLs mentioned above to identify a number of open issues in architectural modeling, and have tried to target those specifically in my research. An early result of this was an ADL that focused on modeling static behavioral properties of architectures [C46]. In the past five years, I have expanded this work in several directions. I have provided a formal semantic basis for component conformance and evolution in an architecture via an architectural type system [C40]; this has enabled several automated analysis capabilities discussed below. I have also expanded the scope of architectural modeling to include, and relate, dynamic behavior and interaction protocol models of components in the form of concurrent state machines [B1, C13, C24, I4]. A recent thrust of my research has been modeling at the architectural level the properties that influence a distributed system's runtime characteristics, specifically availability, latency, and reliability [C4, C8, C9, C35].

**Modeling Application Family Architectures** — Software application families provide promise of significant reuse, and rapid and reliable development of software systems by leveraging their shared properties. At the same time, application families present several interesting challenges from an architectural perspective. Application family architectures must explicitly identify the possible dimensions of system evolution (e.g., via optional components or variant connectors), and ensure that specific application architectures do not deviate from the “reference” family architecture unless explicitly allowed by the architecture. To this end, I have worked on developing a highly extensible ADL [C36] for modeling application family architectures. The ADL was extended with configuration management facilities, which had not been applied at the architectural level prior to this, and then integrated into a software architecture modeling and analysis infrastructure [J1, B3, C26, C30, C33]. The resulting technology, *Mae*, uniquely manages the modeling and evolution of architecture-level artifacts both at system development time and at runtime. *Mae* is detailed in a paper included with this packet [J1].

**Architecture-Level Analysis** — One reason for modeling the architectures of software systems and system families is to enable their early analysis for properties of interest. In the case of application family architectures, certain properties can be assessed at the level of the entire family. Recently, I have collaborated on devising a series of metrics for analyzing the structural quality of application family architectures [C11, C25, I3]; this study has shown that even a highly simplified, partial model can be leveraged to provide useful feedback to system (family) architects. The Mae environment analyzes the conformance of interacting software components in an architecture at the level of static behaviors [J8]; it also incorporates third-party solutions for doing so at the level of dynamic behaviors and interaction protocols [C24]. Mae’s own component-based architecture allows flexible introduction of analysis modules for ensuring an architecture’s conformance to different styles [J1, C46]. Recently, I have begun using Mae’s rich architectural models as the basis of a technique that employs a Hidden Markov Model to estimate system reliability [C4]. Finally, I have worked on devising a series of novel, polynomial-complexity algorithms for analyzing a distributed system’s deployment architecture, and improving the system’s resulting availability and latency by suggesting a re-deployment [C7, C9].

**Research Grants** — The primary sources of support for modeling, analyzing, and measuring application family architectures have been my NSF CAREER [G4], JPL [G8], and Xerox [G10] grants. The work on modeling and assessing system reliability at the architectural level has been supported by the NASA HDCCP [G3] grant. The recent work dealing with architectural aspects of availability and latency in distributed systems has been funded by my NSF ITR [G1] grant; the resulting solutions are currently being adapted and transitioned to the domain of space data systems as part of a recent JPL [G6] grant.

**Ph.D. Students** — The work on architecture modeling and analysis, application families (including the development of Mae), as well as the reliability analysis has been spearheaded by my doctoral student Roshanak Roshandel, with significant contributions from doctoral student Marija Mikic-Rakic. The work on distributed system availability and latency has been led by Marija Mikic-Rakic, who graduated in July 2004, with significant contributions from doctoral student Sam Malek.

### 3. Relating Architecture to Requirements, Design, and Implementation

It is widely acknowledged that an effective architecture forms the necessary basis for a successful software project. At the same time, there are few documented processes and techniques whereby the architecture is arrived at from system requirements and refined into system designs (and, further, implementations). Moreover, many modern systems suffer from architectural “erosion” once they are in operation: changes to the implemented system are not (correctly) reflected in its architectural models, and eventually may even end up violating key architectural decisions. My work has studied these important relationships and has resulted in several techniques for establishing and preserving them. These techniques are referred to as *model connectors*; the paper in which the notion of model connectors is introduced [J4] is one of “Journal of Systems and Software most downloaded articles” published in 2003.

**Architecture and Requirements** — The relationship between the requirements a software system must satisfy and the system’s architecture is not well understood. Little guidance and few methods are available for “architectural discovery”, i.e., consistent refinement of software requirements into an effective architecture that will satisfy those requirements. Part of the challenge stems from the fact that requirements and architectures use different terms and concepts to

capture the model elements relevant to each. My work in this area has resulted in a technique, called *CBSP*, that provides a systematic way of reconciling requirements and architectures [J3, C27, C32, C38]. Specifically, CBSP leverages the basic set of architectural concepts (Components, connectors or *Buses*, configurations or *Systems*, and their *Properties*) to recast and refine the requirements into an intermediate model. This intermediate model leverages the characterization of architectural styles produced as part of the Alfa project discussed above, and is used to arrive at an architectural model that satisfies the requirements.

**Architecture and Design** — My work has directly leveraged the study of UML as a possible ADL, discussed above, to provide a technique for transforming an architectural model into a set of UML-based design models [J4, J7, C29, C31, C42, C43, C44]. When published initially [C43], this was the only technique of its kind. As demonstrated in [J7], a paper included with this packet, the technique is independent of the ADL or style used for modeling the architecture. The insights into the relationship of architecture and design gained from this work have directly helped my work on architectural recovery discussed below.

**Architecture and Implementation** — A key aspect of a software architecture's effectiveness is the ability to preserve architectural decisions in the system's implementation. One way of ensuring this is to automatically generate implementations from architectural models, but very few practical approaches exist to support this. Another way is to restrict the space of implementation decisions, e.g., by imposing a particular implementation framework or middleware platform. I have extensively studied the role of middleware in implementing software architectures, and particularly in implementing software connectors [J6, B2, C21, C39, C41, C45]. These studies have indicated that existing middleware platforms typically fail to support certain key aspects of architecture-based software development, including communication ports, software connectors, explicit system topologies, architectural styles, and so forth. This influenced the development of a family of middleware platforms, referred to as *architectural middleware*, which provide native implementation-level support for architectural concepts [J14, C14, C15, C20, C46]. As further discussed below, I have demonstrated that supporting architectural concepts in architectural middleware does not come at the expense of system efficiency, scalability, or flexibility.

**Architectural Recovery** — Architectural recovery is a process frequently used to cope with architectural erosion whereby the current, "as implemented" architecture of a software system is extracted from the system's implementation. I have developed an evolution-driven, light-weight approach to architectural recovery, called *Focus* [J2, C28, I1]. Focus uses a system's evolution requirements to incrementally recover the system's architecture: it allows engineers to direct their primary attention to the part of the system that is immediately impacted by the desired change; subsequent changes will uncover additional parts of the architecture. Focus is unique in that, in addition to software components, which are the usual target of recovery approaches, it also recovers software connectors and candidate architectural styles. To this end, Focus leverages my study of connectors and styles discussed above. Recently, I have recognized that architectural recovery may be effectively complemented by architectural discovery in stemming architectural erosion [C16, I2]; I am currently pursuing this new research direction.

**Research Grants** — To date, the work on architectural discovery has been supported primarily by my U.S. Army TACOM [G9] grant. The work on architectural refinement into design and implementation, as well as architectural recovery, has been supported by my NSF CAREER [G4] and DARPA DASADA [G7] grants.

**Ph.D. Students** — The work on developing architectural middleware has been spearheaded by my doctoral student Marija Mikic-Rakic (graduated), with significant contributions from Sam Malek and Nikunj Mehta (graduated). The work on architectural recovery (i.e., Focus) has been spearheaded by my doctoral student Vladimir Jakobac.

#### **4. Role of Architecture at Runtime**

Software architecture is the linchpin of a software system’s development. As such, it should play an important role in the system’s execution, as argued in a paper I co-authored on this subject [J10]. This is particularly important in the emerging class of highly distributed, mobile, and embedded systems, possibly running on large numbers of “small” (i.e., resource-constrained) platforms such as handheld devices. I have referred to this class of systems, and the accompanying challenges they impose on software development, as *Prism* (*Programming in the small and many*), both as a convenience and to distinguish it from the traditional software development paradigm of PitL (*Programming in the Large*). Prism has become a significant facet of my research over the past three years. The challenges of Prism have been outlined in three recent papers [J5, J14, C23]. In particular, [J5] appeared in a special issue of IEEE Computer on handheld computing, and was one of only five papers accepted from 87 submissions. My work to date on Prism has studied the impact of the implementation platform on runtime system properties, as well as issues in system deployment and mobility (and the requisite runtime evolution).

**Implementation Platform** — I have hypothesized that an explicit architectural focus will result in effective notations, techniques, and tools for developing and evolving Prism systems. By extension, it should be possible to develop architectural middleware (i.e., middleware that provides direct support for architectural abstractions, as discussed above) for Prism systems that is highly (1) efficient, to minimize the processing, memory, and communication overhead in resource-constrained computing environments; (2) scalable, to support arbitrarily large and widely distributed systems; and (3) extensible, to support a wide range of development and runtime scenarios. To validate this hypothesis, I leveraged my prior work in the area of middleware (see above) to develop *Prism-MW* [J5, C14, C15], an event-based architectural middleware for Prism systems that satisfies the three requirements. Prism-MW couples extensive separation of concerns with several novel optimization techniques described in [C14], a paper included with this packet, to result in a solution that natively supports architectural abstractions, scales to tens of thousands of components and connectors, and is significantly smaller *and* faster than comparable existing solutions. Prism-MW has also been successfully evaluated by two external software development organizations for use in their embedded systems; one of those evaluations has resulted in the ongoing technology transfer effort with JPL. A paper currently in submission [J14] discusses a more recent version of Prism-MW, which supports the implementation of software architectures according to the rules of different styles, while, at the same time, further reducing the middleware’s size and increasing its speed.

**Deployment, Mobility, and Runtime Evolution** — Deploying a software system onto its target hardware hosts is particularly challenging in highly distributed and mobile environments with possibly unstable network connectivity. Effective support for this task requires a model of the system’s planned deployment architecture, the ability to analyze it for key properties, as well as the ability to transfer components to their intended destinations, load them, and start the overall system in a synchronized manner. My work in this area has resulted in a deployment environment, called *Prism-DE* [C22], that performs these tasks using Prism-MW as the underlying implementa-

tion, migration, and execution platform. This approach leverages Prism-MW's extensibility to support unintrusive monitoring [C5] of the running subsystem on each host, and to inform Prism-DE of situations that may affect the system's availability or communication latency, such as unreliable or overloaded network links. In response, Prism-DE may invoke one or more of the algorithms discussed above [C7, C9] to remedy the situation by rapidly calculating a re-deployment architecture for the system. An extensive study of techniques for supporting disconnected operation in distributed systems [J13, C17] indicates that Prism-DE calculates a system's re-deployment faster by several orders of magnitude than comparable approaches. Once the appropriate re-deployment is selected, Prism-MW's event-based nature, explicit software connectors, and extensibility are leveraged to provide support for safely adding, removing, packaging, and transferring software modules in a running system [C8, C9, C15, C34].

**Research Grants** — The work on Prism was initially funded in the context of broader efforts by my DARPA RENES [G12], and then DARPA DASADA [G7] grants. My recent NSF ITR [G1] grant has focused specifically on the issues and challenges of Prism, while the related goals of my U.S. Army TACOM [G9] and JPL [G6] grants have been to transfer the technology resulting from this project into the domains of military ground vehicles and space data systems, respectively. A grant from Intel [G11] has enabled me to equip USC's Embedded Systems Laboratory, which is used in support of my research on Prism and which I am co-directing with USC Professor Gaurav Sukhatme.

**Ph.D. Students** — The work on Prism has been spearheaded by my doctoral student Marija Mikic-Rakic (graduated). Recently, my doctoral student Sam Malek has begun extending and applying this work to decentralized systems. A recent addition to my research group, Christian Mattmann, is a JPL software engineer who has begun assessing the applicability of the Prism technologies to complex space data systems.

## 5. Future Research

The work I have pursued to date has already produced significant results, with a number of actual and several potential impacts. In addition to applying the results of my research in the context of external projects (currently conducted with Boeing and JPL), a number of research threads are still on-going. These on-going threads include

- modeling and analyzing architectural reliability in a compositional manner, such that individual component reliabilities are combined to assess system-wide reliability;
- augmenting static analysis of a system's implementation with analysis of its dynamic behavior to improve architectural recovery;
- combining architectural discovery and recovery to better deal with architectural erosion;
- decoupling a distributed system's model from the analyses performed on that model, in order to investigate aspects of system dependability beyond availability and latency (e.g., security, fault tolerance); and
- leveraging explicit architectural models to optimize routing in publish-subscribe systems deployed over wide-area networks.

In addition, several new research threads are emerging.

One emerging thread deals with the possibility of automatically generating system implementations from Alfa models of style-based software architectures. Since Alfa captures the five key aspects of styles in a small number of recurring primitives that are accompanied with specific composition rules, it is my hypothesis that this information can be used in a manner similar to the

way assembly language instructions are used to implement statements in high-level programming languages. While the potential benefits of such a solution would be significant, a number of challenges will have to be overcome even if the idea proves to be feasible. These challenges will include scalability of the primitive compositions, efficiency of the resulting implementations, and system evolvability and distributability.

Another emerging research thread is the role of software architectures in decentralized systems. The shared underlying assumption of architecture-based approaches to date has been the existence of some sort of architectural model. Such a model will be partial at best, and potentially misleading, in a decentralized system. It is an open research problem how and to what extent one can leverage such a model, and whether “model discovery” can be approached in the same manner as resource discovery has been in modern distributed systems.

The envisioned point of convergence for all of the research threads described in this document is integrated, dependable support for “round trip” architecture-based software engineering, from models of system requirements, applicable architectural styles, and reference (i.e., application family) architectures; to application architectures, designs, and implementations; to deployed, running, and evolving systems; and back. While I am sure to face significant challenges in accomplishing this objective, I believe that my efforts to date, as well as the planned further efforts, provide me with a unique platform from which to pursue it.