Statement of Research Interests

Rean Griffith

Overall Research Interests and Goals

My goal is to improve the manageability of computing systems. Coping with increasing complexity is one of the major challenges facing the computing industry today, and for the foreseeable future. Complexity impacts many of the activities associated with realizing computing systems including, but not limited to, aspects of system-construction, system-testing, system-deployment, system-configuration, system-operation, and system-maintenance. As a result, it affects a wide cross-section of stakeholders (developers, administrators, end-users etc.) and manifests itself in different ways: outages, slowdowns, hangs, mis-configurations, complex problem-diagnosis and problem-resolution and other reliability, availability and serviceability (RAS) issues. Our reliance on computing systems and the role they play in the provision of financial, health, communication and manufacturing services necessitates that we devise and evaluate tools and techniques that will allow us to better understand and manage the systems we are building and have built.

Whereas, my past and current research projects are unified by the goal of addressing system complexity, the details and scope of the individual projects themselves are considerably varied.

Early work on DISCUS (Decentralized Information Spaces for Composition and Unification of Web Services) addressed issues concerned with exposing contemporary and legacy systems as services and effectively aggregating services across organizational boundaries/trust domains to solve a given problem. DISCUS was a proof-of-concept realization of NICCI (Network-centric Infrastructure for Command Control and Intelligence) a prospective DARPA program (circa 2002) concerned with coalition forces in crisis situations. My role in DISCUS was to design and implement the middleware component (Gatekeeper) responsible for negotiating for services across trust domains and orchestrating the workflow involving these aggregated services.

Interest in the continuous validation of software systems led to a short project working on Kinesthetics eXtreme (KX). KX was originally developed under the DARPA DASADA (Dynamic Assembly for System Adaptability, Dependability and Assurance) project and was concerned with the design and implementation of an Externalized Monitoring and Adaptation Infrastructure for distributed legacy systems. The goal for KX was to propose a standards-based approach to runtime monitoring (continual validation) of component-based systems. KX was used to retrofit adaptive/autonomic behavior onto legacy systems to facilitate continual validation. The KX project involved the development and integration of a number of technologies including: the use of software probes to collect information (events) from systems, the development and integration of content-based publish-subscribe architectures (Siena, Elvin) used as an event transport and the development of mobile-agent technologies (Worklets) used to effect adaptations in the systems being monitored. My role in the KX project was to re-implement portions of the original Event Distiller component used to aggregate data from distributed monitoring probes and trigger adaptations in the systems being monitored.

Continued work on retrofitting autonomic (specifically self-healing)/self-management mechanisms onto legacy systems led the development of Kheiron. Kheiron is a framework used to transparently and dynamically effect adaptations in applications while they execute, with low overhead. Kheiron uses the existing facilities of execution environments to interact with and adapt running systems. Using a combination of bytecode rewriting (and binary rewriting) Kheiron is able to adapt running .NET applications, Java applications and compiled C programs running on Linux x86. In past experiments Kheiron has been used to: monitor program execution, perform hot-swaps of key system components, insert and remove instrumentation and add and remove code that effects dynamic re-configurations. Whereas the development of Kheiron reached a point where it was able to interact with both managed applications (e.g. .NET and Java) and unmanaged applications (e.g. compiled C programs/ELF binaries on Linux), an open question remained: How could we show that systems using Kheiron (or other adaptation facilities) to

add/retrofit autonomic/self-healing behaviors were quantitatively better than their non-self-healing/legacy counterparts?

Answering this evaluation question is the focus of my thesis research. My proposed solution identifies reliability, availability and serviceability (RAS) as reasonable metric-categories for quantitatively evaluating self-healing as well as non-self-healing systems given the extra-functional demands placed on the computing systems we depend on. My thesis is focused on developing practical tools and analytical techniques that allow us to study and evaluate the (expected or actual) reliability, availability and serviceability characteristics of software systems at design-time or post-deployment. Further, the runtime adaptation capabilities of Kheiron are now used to build fault-injection tools that are used to exercise (or highlight the lack of) any RAS-enhancing mechanisms a system might (or should) have.

Despite being involved in research projects which involved: devising architectures, components, middleware, design methodologies for building new systems and retrofitting legacy systems, my approach to research has four common characteristics:

1. Wide applicability – complexity impacts the manageability of many classes of systems, including, but not limited to: distributed systems, high-performance computing systems, self-managing/autonomic systems and networks, real-time systems, embedded systems, N-tier web applications, standalone applications, virtual machines, sensor-networks and mobile networks. This allows me to collaborate with research colleagues in other areas of computer science and as well as a variety of industry partners. In the past I have collaborated with Joseph L. Hellerstein (IBM) on the application of Control Theory as a building block for self-managing systems. I am currently involved in separate collaborations with Sun Microsystems, Stacksafe Inc. and a Ph.D. Student (Javier Alonso, advised by Prof. Jordi Torres) at the Technical University of Catalonia (UPC)/Barcelona Super-Computing Center. These collaborations are concerned with evaluating the efficacy of the Fault Management Architecture (FMA) of the OpenSolaris operating system, the reliability, availability and serviceability (RAS) benchmarking of N-tier web-applications, and the RAS-evaluation of Virtual-Machine based self-healing techniques for autonomic systems respectively.
2. Experimental – driven by hands-on tool and/or system building and instrumentation. A major part of my work involves collecting data on, and manipulating the internals of, software systems in execution. Building the tools, experimental prototypes or experimental testbeds requires (at times) low-level interactions with (state-of-the-art or legacy) execution environments and software and hardware components of interest. As a result I have gained tremendous insight into the mechanics of sophisticated hardware and software platforms.
3. Quantitative – focused on identifying metrics that can be used to evaluate and compare systems according to one or more facets of manageability. The metrics of interest are influenced by considering one or more stakeholder perspectives and the class of application of interest. In conducting these evaluations and comparisons, performance is only one metric of interest as opposed to being the primary metric of interest. Faster does not necessarily equate to "more manageable", "more reliable", "better at self-healing" etc. as a result I seek to identify other criteria that can be used to quantitatively compare systems along facets of manageability.
4. Interdisciplinary – evaluating and comparing the manageability of computing systems may require adapting approaches, tools and analytical techniques from other branches of computer science and other engineering or non-engineering disciplines. Examples of areas I have previously drawn on include, reliability modeling, control theory, computer architecture, software engineering and programming languages and compilers. However, opportunities remain for employing other mathematical modeling techniques, machine-learning, insights from human-computer interaction and analytical techniques from economics and finance as part of the evaluation and comparison process.

My future research will retain these characteristics. I will continue to validate ideas by constructing systems, tools and testbeds, and I will draw on my interdisciplinary background to identify and adapt tools and techniques as necessary.

Thesis Summary and Contributions

My current research exemplifies these four characteristics. My thesis designs a reliability, availability and serviceability (RAS) evaluation methodology and implements a RAS-benchmark for N-tier web applications.

Whereas I chose N-tier web-applications for their ubiquity and the fact that they are composed of non-trivial components (web-server, application server, database server and one or more operating system instances), the motivation for the thesis stems from renewed interest in developing computing systems that meet additional non-functional requirements such as reliability, high availability and ease-of-management/self-management (serviceability).

However, to reason about trade-offs between RAS-enhancing mechanisms or to evaluate these mechanisms and their impact we need something other than performance metrics. Whereas performance metrics are suitable for studying the feasibility of having RAS-enhancing mechanisms activated, i.e. to demonstrate that the system provides "acceptable" performance with these mechanisms enabled – allowing us to draw conclusions about whether the performance delivered by the system with these mechanisms enabled precludes its usage – the resulting performance numbers convey little about the efficacy of the mechanisms.

Performance measures do not allow us to analyze the expected or actual impact (beyond system overheads) of individual or combined RAS-enhancing mechanisms on the system's operation. They are inadequate for comparing the efficacy of individual or combined RAS-enhancing mechanisms, discussing trade-offs between mechanisms, evaluating different styles of mechanisms (reactive vs. preventative vs. proactive) or reasoning about the composition of multiple mechanisms. In essence, performance metrics limit the scope and depth of analysis that can be performed on systems possessing (or considering the inclusion of) RAS-enhancing mechanisms.

Reasoning about the RAS-capabilities of the systems of today or the self-* systems of tomorrow also involves addressing three evaluation-related challenges. First, developing (or identifying) practical fault-injection tools that can be used to study the failure behavior of computing systems and exercise any (remediation) mechanisms the system has available for mitigating or resolving problems. Second, identifying techniques that can be used to quantify RAS-deficiencies in computing systems and reason about the efficacy of individual or combined RAS-enhancing mechanisms (at design-time or after system deployment). Third, developing an evaluation methodology that can be used to objectively compare systems based on the (expected or actual) benefits of RAS-enhancing mechanisms.

My thesis addresses these three challenges by introducing the 7U-Evaluation Methodology, a complementary approach to traditional performance-centric evaluations that identifies criteria for comparing and analyzing existing (or yet-to-be-added) RAS-enhancing mechanisms, is able to evaluate and reason about combinations of mechanisms, exposes under-performing mechanisms and highlights the lack of mechanisms in a rigorous, objective and quantitative manner. By using the 7U-Evaluation Method to evaluate the RAS-capabilities of real-world computing systems, my thesis makes three contributions.

First, fault-injection techniques and tools that facilitate "in-situ" and "in-vivo" interactions with computing systems. "In-situ" interactions allow us to study the failure-behavior of a computing system in its deployed environment while "in-vivo" interactions allow us to inject faults into running systems. Both techniques offer advantages for studying the failure-behavior of computing

systems and are exemplified by the fault-injection tools built using the Kheiron tool-suite I developed. Kheiron tools interact dynamically, transparently with applications running in contemporary managed execution environments e.g. Sun Microsystems' Java Virtual Machine (JVM) and Microsoft's Common Language Runtime (CLR), as well as unmanaged execution environments, the operating system (e.g. Linux) and the raw processor (e.g. x86). Using this foundation I am able to inject faults into a variety of applications and application-components in execution to study their responses (or lack thereof).

Second, analytical tools that can be used construct mathematical models (RAS-models) to evaluate and quantify RAS-capabilities. Quantifying fault-impacts in terms of RAS-metrics involves identifying and measuring the facets of reliability, availability and serviceability that vary when faults occur. Fortunately, there are many facets of reliability, availability and serviceability that can, and have been used in the past, to quantify fault-impacts including, but not limited to: frequency of service interruptions or outages, yearly downtime and its associated "costs" (time and money spent on restoring complete or partial service, time and money lost due to system unavailability, end-user downtime etc.), meantime to system breakdown, the number of servicing visits, the frequency of servicing visits, the ability to meet SLA targets, the ability to meet production targets and the ability to avoid or mitigate production slowdowns and system stability. Further, there are a number of mathematical tools that can be used to model individual or combined RAS-enhancing mechanisms and quantify their capabilities in terms of the metrics of interest. In my thesis I use Markov chains, Markov Reward Networks and Control Theory models to reason quantitatively about expected and actual system/component behavior.

Finally, the results and insights gained from conducting fault-injection experiments on real-world systems and modeling the system responses (or lack thereof) using RAS-models are presented. In conducting 7U Evaluations of real-world systems and components, I demonstrate the utility of RAS-models as design-time or post-deployment tools given their ability to reason about existing or yet-to-be added RAS-enhancing mechanisms. Further, I identify key characteristic of the RAS-evaluation process – the ability to study the failure behavior of systems, the ability to evaluate individual or combinations of mechanisms, the ability to evaluate various styles of mechanisms, the ability to quantify the impacts of imperfect recovery and the ability to identify under-performing mechanisms. Finally, my thesis highlights the similarities and differences between traditional performance-oriented evaluations and RAS-oriented evaluations and outlines a general framework for conducting RAS-evaluations.

Future Directions

My thesis has spawned a number of possible avenues for future research. First, the work done on runtime instrumentation and adaptation of systems has led to questions about the facilities that need to be available to support adaptivity in systems and the safety guarantees that can be given with respect to adaptations.

In my thesis, facilities of contemporary execution environments were used to perform adaptations (instrumentation and fault-injection) on applications in execution, however I was only able to give "weak" safety guarantees based on empirical results. Further, whereas I was able to show that it is comparatively easier to perform adaptations in managed execution environments than in unmanaged execution environments by virtue of the metadata available in managed execution units (bytecode in classfiles and assemblies) and the services provided by the execution environment for program tracing and metadata creation/modification, there were no other guarantees besides ensuring that bytecode instructions are well-formed and bytecode modifications do not violate type safety or existing security policy rules.
The ability to manipulate programs in execution is a powerful yet risky facility. However, stakeholders will be wary of using runtime adaptation facilities in production systems without stronger guarantees on their safety. The form and the degree to which we can express and codify such guarantees is still an open question, but the increasing sophistication of system-construction

tools (high-level languages, modeling tools etc.) and application execution environments (managed execution environments e.g. the JVM and CLR and unmanaged execution environments e.g. operating systems, processors, Xen, VMWare, hypervisors etc.) may provide insights into additional support for realizing adaptive systems.

A second research direction is concerned with developing non-traditional benchmarks for other classes of systems. There are a number of other areas in computer science currently investigating what self-managing behavior means for them. Examples include networking and network management systems, media-delivery platforms and grid-computing environments. Part of this process will involve establishing some criteria for quantifying the improvements resulting from the inclusion of self-managing technologies into the systems they have today and plan to build tomorrow. Ideas adapted from my thesis and other work on non-traditional benchmarking efforts can have some applications here.

Outside of my thesis work, I am interested in the incorporation of semantic information into programming languages, execution units (binaries) and runtime environments to improve system monitoring and problem diagnosis. The increased sophistication of debugging and profiling tools has allowed users, developers and researchers to gather detailed data on the current execution steps of one or more software components, including parts of the underlying operating system. However, tracking the logical progress of a large and/or distributed application via the flood of method invocation and component transition data can be difficult if not daunting. Recent advances in programming language constructs (e.g. annotations and language extensions), compiler technologies (semantically-richer intermediate-representations of programs) and sophisticated runtime environments have made it easier to embed and access metadata about program elements, component dependencies and component attributes in the execution units (binaries, linked libraries) of an application. These advances are an excellent platform for continuing research into providing semantically-rich information on program execution and intended operation. I will work on including and using this semantic information to build advanced program analysis tools aimed at improving the monitoring, diagnosis and by extension predictability of software systems.

Long Term Goals

Motivated by complexity of software systems (especially distributed software systems) I am also interested in system-design theory, specifically the development of techniques and tools that improve the predictability of systems. Traditionally, advances in system-design theory (system architectures, design patterns, specification tools, modeling tools, synthesis tools, programming language improvements etc.) have slowly made their way into areas of industry, my practice-oriented approach will help bridge the gap between system-design theory and the system-construction practice. This focus brings together hands-on (at times low-level) systems research and interdisciplinary approaches for constructing various classes of systems and establishing criteria for evaluating their efficacy. Striking a balance between the two will expose me to a diverse set of research opportunities and allow me to interact with research colleagues and students inside and outside of Computer Science while investigating new and innovative ways to address the complexity inherent in software systems.