Dependable autonomous systems = Knowing well what to do + Knowing how to do it well

Dr. Nicolas Rouquette
Jet Propulsion Laboratory
4800 Oak Grove Dr, M/S 301-270
Pasadena, CA 91109
nicolas.rouquette@jpl.nasa.gov

Dr. Nenad Medvidovic
Computer Science Department
University of Southern California
Los Angeles, CA 90089-0781
neno@usc.edu

Dr. David Garlan
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3891
garlan@cs.cmu.edu

This position paper reflects upon experiences and lessons learned from several spacecraft projects at JPL and JPL's recent collaboration with software architecture researchers from USC and CMU. In the last 15 years, dependability at JPL has evolved in stages. Cassini is the textbook example of a process-heavy project that produced a class-A spacecraft ("a Rolls-Royce for space"). The "faster/better/cheaper" mandates of Mars Pathfinder and Deep Space One (DS1) gave rise to software pragmatism ("fly what you test & test what you fly"). The fault-protection work on DS1 pushed the software process to an extreme adoption of "test like hell". For a fixed budget, the idea was to drive software development costs towards zero using model-based code generation technology and spend all remaining resources on the best and brightest system engineers to analyze, debate, test and operate the best spacecraft fault protection we could afford using statechart models. The DS1 Remote Agent pushed the technology envelope of autonomous systems capable of deliberating, planning, scheduling and troubleshooting the spacecraft's mission and activities. These projects solved their own dependability computing challenges with various combinations of engineering, ingenuity, management, tools, and test.

Today, the MDS project at JPL represents the fulcrum of a drive towards the routine development of dependable space missions. Several ingredients helped shape MDS into what we believe is our best shot at a product-line approach to JPL's future space missions.

First, we traditionally limited ourselves to view software in terms of abstraction levels. When our abstractions were inadequate, we reverted to the next lower level such as programming language statements. Analyzing the dependability of software systems at that level is inadequate for thinking about complex spacecraft systems. Researchers in verification/validation (V&V) have known this for years but the adoption of their practices has been slow to catch on. To resolve this abstraction mismatch, we need abstractions from multiple architectural styles that yield a rich architectural landscape on which we can try to operate while, at the same time, ensuring a high degree of dependability. The MDS project represents a marriage of heterogeneous architectural styles; a research problem jointly researched with David Garlan. In particular, MDS combines a domain-specific *software* architecture (DSSA) centered on the concept of state with a component/connector view of software architecture.

We already know a very useful and effective domain abstraction for talking about space systems and missions: *state*. In MDS is a novel architectural approach to state analysis in that it allows arbitrary notions of state to be reified as entities that the architecture can use and operate on. This is a fundamental property to design autonomous systems that reason, deliberate, and control their own state to achieve goals expressed as

constraints on state properties. However, the DS1 Remote Agent experiment showed us a single state-centered architectural style is not enough. Although it had extensive reasoning capabilities, the experiment had to be aborted when two of the Remote Agent subsystems ran into a resource deadlock. This is a classic example of architecture mismatch that V&V researchers managed to demonstrate. However, the Remote Agent had no cognizance of its own execution architecture to detect it. MDS has an explicit component/connector software architecture, the second dimension of abstraction we need.

In MDS, we seek to extend the "test what you fly/fly what you test" sense of pragmatism to our bi-dimensional engineering approach. For example, we are researching not only ways to prototype components and connectors using a statechart formalism but also research extensions of architecture description languages that incorporate features of state-based analysis, for example to describe connector-mediated interaction protocols.

Traditionally, notions of conceptual software architecture with components, connectors and interfaces have been limited to analysis, not explicit implementation with poor first-class software representation. For example, traditional notions of software initialization levels are not used to determine how far back to restart in the presence of faults: common wisdom has it that optimizations and cached references create interdependencies too difficult to undo and re-establish with great confidence. We believe a conceptual architecture with first-class components, connectors and interfaces is required to bring in additional tools for designing fault isolation and recovery techniques that complement an autonomous systems' model-based reasoning capabilities for analyzing faults in the state-based domain. There are several challenges to this approach.

The first challenge attacks the principle that all interactions among components occur via connectors. How far deep do we reify all communication with connectors? We run into limitations due to our ability to define these software architecture abstractions in terms of traditional software language constructs, mechanisms and patterns. Performance considerations play an important role for optimizing dynamic interactions that are too infrequent or too small to incur the burden of connection establishment. These limitations jeopardize the integrity of the architecture and introduce the risk of adverse interactions like those the remote agent encountered.

How do we prove to a decision-making manager that a complex product family like MDS will yield dependable systems? We intend to leverage the DSSA's goal-based architecture and comprehensive reification of components, connectors and interfaces to confer the system self-awareness of its capabilities, performance and actual competency. Cognizant failure, a common trait of autonomous systems, is a key enabler to raise the system's self-awareness about its problem-solving performance traceable to the specific contributions and roles the involved elements of the architecture played. This represents an important shift of focus for the artificial intelligence reasoning techniques where the software architecture becomes part of the model reasoned about. In summary, dependability and autonomous systems go hand in hand: They must have full cognizance of their dependability in terms of their domain of expertise and of their internal software architecture. Besides enabling dependable computing, component software architectures and related integration standards will enable better engineering processes for building complex avionics software as dependable assemblies of vendor-supplied components.

Acknowledgements

The portion of research by Nicolas Rouquette described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

The portion of research by Neno Medvidovic described in this paper was carried out at the University of Southern California, under a contract with the National Science Foundation and a contract with the National Aeronautics and Space Administration through the Jet Propulsion Laboratory, California Institute of Technology.

The portion of research by David Garlan described in this paper was carried out at Carnegie Mellon University, under a contract with the Defense Advanced Research Projects Agency.

Biographies

Nicolas Rouquette is a Senior Computer Scientist at the Jet Propulsion Laboratory. He has experience in designing and mentoring spacecraft fault protection systems using COTS model-based code generation technologies. His current work and research interests focus on organizing heterogeneous software architecture styles, frameworks and patterns around the central notions of first class components, connectors and interfaces.

Nenad Medvidovic is an Assistant Professor in the Computer Science Department at the University of Southern California and is a faculty member of the USC Center for Software Engineering (CSE). His research interests focus on software architecture as a key to developing mechanisms for engineering reliable, flexible, large-scale software. His research project, SAAGE (Software Architecture, Analysis, Generation, and Evolution), expands the traditional notions of evolution, such as modularity and typing, for use in architectures, and introduces explicit, flexible connectors.

David Garlan is an Associate Professor of Computer Science at Carnegie Mellon University. His research interests include software engineering, with a focus on software architecture, pervasive computing, formal specifications, software development environments. He co-authored with Mary Shaw one of the first modern textbooks on software architecture.