PERGAMON

Available online at www.sciencedirect.com

SCIENCE @ DIRECT®

Acta Astronautica 58 (2006) 648–661

ACTA ASTRONAUTICA

www.elsevier.com/locate/actaastro

Generating requirements for complex embedded systems using State Analysis

Michel D. Ingham*, Robert D. Rasmussen, Matthew B. Bennett, Alex C. Moncada

Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive M/S 301-225, Pasadena, CA 91109, USA

Received 21 September 2005; accepted 3 January 2006
Available online 4 May 2006

Abstract

It has become clear that spacecraft system complexity is reaching a threshold where customary methods of control are no longer affordable or sufficiently reliable. At the heart of this problem are the conventional approaches to systems and software engineering based on subsystem-level functional decomposition, which fail to scale in the tangled web of interactions typically encountered in complex spacecraft designs. Furthermore, there is a fundamental gap between the requirements on software specified by systems engineers and the implementation of these requirements by software engineers. Software engineers must perform the translation of requirements into software code, hoping to accurately capture the systems engineer's understanding of the system behavior, which is not always explicitly specified. This gap opens up the possibility for misinterpretation of the systems engineer's intent, potentially leading to software errors. This problem is addressed by a systems engineering methodology called State Analysis, which provides a process for capturing system and software requirements in the form of explicit models. This paper describes how requirements for complex aerospace systems can be developed using State Analysis, using representative spacecraft examples.

© 2006 International Astronautical Federation. Published by Elsevier Ltd. All rights reserved.

1. Introduction

As the challenges of space missions have grown over time, we have seen a steady trend toward greater automation, with a growing portion assumed by the spacecraft. This trend is accelerating rapidly, spurred by mounting complexity in mission objectives and the systems required to achieve them. In fact, the advent of truly self-directed space robots is not just an imminent possibility, but an economic necessity, if we are to continue our progress into space.

What is clear now, however, is that spacecraft design is reaching a threshold of complexity where customary methods of control are no longer affordable or sufficiently reliable. At the heart of this problem are the conventional approaches to systems and software engineering based on subsystem-level functional decomposition, which fail to scale in the tangled web of interactions typically encountered in complex spacecraft designs. A straightforward extrapolation of past methods has neither the conceptual reach nor the analytical depth to address the challenges associated with future space exploration objectives.

Furthermore, there is a fundamental gap between the requirements on software specified by systems engineers and the implementation of these requirements by software engineers. Software engineers must perform the translation of requirements into software code,

* Corresponding author.
E-mail addresses: michel.d.ingham@jpl.nasa.govak (M.D. Ingham), robert.d.rasmussen@jpl.nasa.gov (R.D. Rasmussen), matthew.b.bennett@jpl.nasa.gov (M.B. Bennett), alex.c.moncada@jpl.nasa.gov (A.C. Moncada).

hoping to accurately capture the systems engineer's understanding of the system behavior, which is not always explicitly specified. This gap opens up the possibility for misinterpretation of the systems engineer's intent, potentially leading to software errors.

In this paper, we describe a novel systems engineering methodology, called State Analysis, which addresses these challenges by asserting the following basic principles:

- Control subsumes all aspects of system operation. It can be understood and exercised intelligently only through models of the system under control. Therefore, a clear distinction must be made between the *control system* and the *system under control*.
- Models of the system under control must be explicitly identified and used in a way that assures consensus among systems engineers. Understanding state is fundamental to successful modeling. Everything we need to know and everything we want to do can be expressed in terms of the state of the system under control.
- The manner in which models inform software design and operation should be direct, requiring minimal translation.

State Analysis improves on the current state-of-the-practice by producing requirements on system and software design in the form of explicit models of system behavior, and by defining a state-based architecture for the control system. It provides a common language for systems and software engineers to communicate, and thus bridges the traditional gap between software requirements and software implementation. The State Analysis methodology is complemented by a database tool that facilitates model-based software requirements capture.

1.1. Paper outline

In this paper, we discuss the state-based control architecture that provides the framework for State Analysis (Section 2), we emphasize the central notion of state, which lies at the core of the architecture (Section 3), we present the process of capturing requirements on the system under control in the form of models (Section 4), and we illustrate how these models are used in the design of a control system (Section 5). We then discuss the database tool used for documenting the models and requirements (Section 6). Finally, we describe the Mission Data System (MDS), a modular multi-mission software framework that leverages the State Analysis methodology (Section 7).

2. State-based control architecture

State Analysis provides a uniform, methodical, and rigorous approach for:

- discovering, characterizing, representing, and documenting the states of a system;
- modeling the behavior of states and relationships among them, including information about hardware interfaces and operation;
- capturing the mission objectives in detailed scenarios motivated by operator intent;
- keeping track of system constraints and operating rules; and
- describing the methods by which objectives will be achieved.

For each of these design aspects, there is a simple but strict structure within which it is defined: the state-based control architecture (also known as the "Control Diamond", see Fig. 1).

The architecture has the following key features [1]:

- *State is explicit*. The full knowledge of the state of the system under control is represented in a collection of state variables. We discuss the representation of state in more detail in Section 3.
- *State estimation is separate from state control*. Estimation and control are coupled only through state variables. Keeping these two tasks separate promotes objective assessment of system state, ensures consistent use of state across the system, simplifies the design, promotes modularity, and facilitates implementation in software.
- *Hardware adapters provide the sole interface between the system under control and the control system*. They form the boundary of our state architecture, provide all the measurement and command abstractions used for control and estimation, and are responsible for translating and managing raw hardware input and output.
- *Models are ubiquitous throughout the architecture*. Models are used both for execution (estimating and controlling state) and higher-level planning (e.g., resource management). State Analysis requires that the models be documented explicitly, in whatever form is most convenient for the given application. In Section 4, we describe our process for capturing these models.
- *The architecture emphasizes goal-directed closed-loop operation*. Instead of specifying desired behavior in terms of low-level open-loop commands,

Download English Version:

https://daneshyari.com/en/article/1717269

Download Persian Version:

https://daneshyari.com/article/1717269

Daneshyari.com