Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computer Networks 49 (2005) 689–706
Computer Networks
www.elsevier.com/locate/comnet

Developing safety-critical real-time systems with SDL design patterns and components

Ingmar Fliege *, Alexander Geraldy, Reinhard Gotzhein, Thomas Kuhn, Christian Webel

Computer Science Department, University of Kaiserslautern, Postfach 3049, D-67653 Kaiserslautern, Germany

Available online 1 June 2005

Abstract

SDL is a system design language for the development of distributed systems, including real-time systems. In this paper, we apply SDL to capture design solutions found in safety-critical real-time systems. In particular, we present a methodology to augment system safety step-by-step, and systematically define and apply reusable design solutions for safety-critical real-time systems, expressed as SDL design patterns and components. These solutions can be added to real-time system designs, to protect against certain types of system failures. We illustrate the approach by the definition of reusable solutions for the detection of fail-silent nodes—a watchdog and a heartbeat—and their application to a distributed airship flight-control system.

Keywords: Distributed systems engineering; SDL; Reuse; Design pattern; Design component; Reliability; Safety; Real-time system

1. Introduction

A *real-time system* is a reactive system in which the correctness of the system behaviour depends not only on the correct ordering of events, but also on their occurrence in time [1]. It is called *safety-critical*, if its failure may cause harm to humans, or if the cost of failure may be high. Examples of safety-critical real-time systems are flight-control systems of aircraft, or intelligent brake systems for cars.

Safety-critical systems are required to exhibit extremely low rates of critical failures, which are in the order of 10^{-9} failures/hour.[1] To achieve these failure rates, a variety of reliability measures such as hardware redundancy, time redundancy,

* Corresponding author.
E-mail addresses: fliege@informatik.uni-kl.de (I. Fliege), geraldy@informatik.uni-kl.de (A. Geraldy), gotzhein@informatik.uni-kl.de (R. Gotzhein), kuhn@informatik.uni-kl.de (T. Kuhn), webel@informatik.uni-kl.de (C. Webel).

[1] Thus, safety is reliability w.r.t. critical failures [1].

1389-1286/$ - see front matter
doi:10.1016/j.comnet.2005.05.003

software diversity, or recovery can be combined. Also, the functionality of a system may be reduced as a final consequence of a failure by guiding the system to a fail-operational (on aircraft) or fail-safe (on trains) state.

A real-time system may be distributed, consisting of a set of interacting real-time components. In [1], arguments in favour of the distributed approach are given, including that of compositionality. In a compositional architecture, system properties are determined by component properties. If components are well-specified, implemented, and validated, this will augment the overall system quality and thus its reliability. Even more, if well-proven solutions for recurring development problems of safety-critical systems can be isolated, their reuse will improve both productivity and quality of system development.

Due to the nature of a real-time system, its specification comprises both functional and temporal constraints. Furthermore, a distributed approach is advocated (see [1]). A formal design language that explicitly addresses these aspects is SDL, the specification and description language [2]. In [3], the semantics of SDL including the notion of real-time has been formally defined, and forms part of the SDL language definition. Though originating from the telecommunications domain, SDL has successfully proven its broader applicability [4]. Also, the aforementioned reuse concepts have been instantiated in a approach based on SDL [5–7]. Therefore, we have chosen SDL as the design language for developing safety-critical real-time systems.

In this article, we present an engineering approach to stepwise augment the reliability of safety-critical real-time systems by reusable solutions. In Section 2, we survey related work. Section 3 introduces the methodology for the incremental development of safety-critical real-time systems. In Section 4, we systematically derive specific SDL design patterns and components as generic, reusable solutions, to be added to real-time systems to protect against fail-silent nodes. We then elaborate on the use of patterns versus components (Section 5). In Section 6, we present an application of these patterns and components to augment the reliability of a distributed airship flight-control system. The article ends with a summary and an outlook (Section 7).

2. Related work

Reuse has been thoroughly studied in software engineering, and has led to the distinction of three main reuse concepts: components, frameworks, and patterns. A *component* can be characterized as a self-contained ready-to-use building block, which is selected from a component library, and composed. A *framework* is the skeleton of a system, to be adapted by the system developer. A *pattern* describes a generic solution for recurring problems, to be customized for a particular context. For each of these reuse concepts, extensive results have been published. In this survey of related work, we constrain ourselves to results on reuse in the context of SDL and in reliability engineering.

In [7], the structuring unit *micro protocol* has been identified and applied to SDL designs. A *micro protocol* is a communication protocol with a single (distributed) functionality and the required protocol collaboration. A functionality (such as flow control, loss control, QoS monitoring) is a single aspect of internal system behaviour that may be distributed among a set of system agents, with causality relationships between single events. In [8], a fine-grained development process, together with a generic micro protocol framework, is presented. From a reuse viewpoint, micro protocols are classified as design components, which are selected from a library and composed.

In [5], *SDL frameworks* are presented. To define frameworks, the capability of SDL to encapsulate object structures and default behaviour in frameworks defined by SDL system types is exploited. Specific applications can then be obtained by defining subtypes of framework system types, and by redefining virtual types. Development and application of SDL frameworks is integrated into TIMe, the integrated method [9], which is a comprehensive development methodology that uses