Contents lists available at ScienceDirect

Fusion Engineering and Design

journal homepage: www.elsevier.com/locate/fusengdes

Software fault detection and recovery in critical real-time systems: An approach based on loose coupling

Pekka Alho*, Jouni Mattila

Department of Intelligent Hydraulics and Automation, Tampere University of Technology, Finland

HIGHLIGHTS

- We analyze fault tolerance in mission-critical real-time systems.
- Decoupled architectural model can be used to implement fault tolerance.
- Prototype implementation for remote handling control system and service manager.
- Recovery from transient faults by restarting services.

ARTICLE INFO

Article history:
Received 30 August 2013
Received in revised form 7 April 2014
Accepted 22 April 2014
Available online 14 May 2014

Keywords:
ITER
Remote handling
Software
Fault tolerance
Dependability
Real-time

ABSTRACT

Remote handling (RH) systems are used to inspect, make changes to, and maintain components in the ITER machine and as such are an example of mission-critical system. Failure in a critical system may cause damage, significant financial losses and loss of experiment runtime, making dependability one of their most important properties. However, even if the software for RH control systems has been developed using best practices, the system might still fail due to undetected faults (bugs), hardware failures, etc. Critical systems therefore need capability to tolerate faults and resume operation after their occurrence. However, design of effective fault detection and recovery mechanisms poses a challenge due to timeliness requirements, growth in scale, and complex interactions. In this paper we evaluate effectiveness of service-oriented architectural approach to fault tolerance in mission-critical real-time systems. We use a prototype implementation for service management with an experimental RH control system and industrial manipulator. The fault tolerance is based on using the high level of decoupling between services to recover from transient faults by service restarts. In case the recovery process is not successful, the system can still be used if the fault was not in a critical software module.

1. Introduction

Remote handling (RH) systems are used to inspect, make changes to, and maintain components in the ITER machine. Failure in a mission-critical system like RH may cause damage and, perhaps even more significantly, loss of experiment runtime, therefore making dependability one of its most important properties. However, even if the software for the RH system has been developed using valid development processes, the system might still fail due to undetected faults, hardware failures, etc. Critical systems therefore need to be able to resume operation after faults have occurred, but design of effective fault detection and recovery mechanisms poses a challenge. This is due to timeliness requirements combined with growth in scale and complex dynamic interactions in RH systems and embedded systems in general.

Several programming languages and frameworks, e.g. Erlang or OSGi for Java, support use of decoupled architectural models that can be used to implement fault tolerance solutions and dynamic loading of software modules, but these approaches are typically used in non-critical applications that do not have requirements for deterministic response times. In this paper we evaluate effectiveness of the decoupled architectural approach in mission-critical real-time systems using an experimental RH control system for an industrial manipulator. The control system is based on a real-time service oriented architecture (RTSOA) that we have introduced and evaluated in [1]. Services (i.e. the applications that participate in the control of the manipulator) are managed by a prototype

* Corresponding author. Tel.: +358 505375726.
E-mail address: pekka.alho@tut.fi (P. Alho).

http://dx.doi.org/10.1016/j.fusengdes.2014.04.050
0920-3796/© 2014 Elsevier B.V. All rights reserved.

service manager that is used to detect faults and initiate recovery processes.

The RH control system consists of several heterogeneous subsystems, including equipment controller (EC), virtual reality (VR) and operations management system (OMS), specified in the ITER RH control system handbook [2]. This kind of cooperation of several networked computational units is typical for the field of cyber-physical systems (CPS), featuring a tight coordination between computational and physical elements of the system. CPS research aims to improve interoperability and openness between networked controllers to produce more intelligent applications.

2. Background

Fault tolerance means avoiding service failures in the presence of faults, and consists of error detection and recovery [3]. However, recovery can introduce unpredictable delays that might be in conflict with the predictability requirements of real-time systems. A typical approach for real-time fault tolerance is to use two or more diverse versions of software. Such an approach is suitable, e.g. in aviation, where the scope of critical systems is limited, and the cost of creating multi-version software is distributed over a large number of aircraft [4]. However, for large and complex one-off software systems, such as RH, use of multi-version techniques is difficult to justify.

Key challenges for fault tolerance in RH systems include recovery of state data, reliable detection of faults, fault recovery that supports real-time requirements, and ensuring reliability of end-to-end service chains. Safety of the system relies heavily on reasoning about consequences of faults, which is an important open research area due to the complex and stochastic nature characteristic for CPS.

Previous research on real-time fault tolerance has focused largely on redundancy-based solutions and reconfiguration. Gonzales et al. use adaptive management of redundancy to assure reliability of critical modules by allocating as much redundancy to less critical modules as could be afforded, thus gracefully reducing their resource requirements [5]. Assured reconfiguration in case of failures is used in [6]. This allows the primary function to fail and then reconfigure to some simpler function – reconfiguration of the system is a critical part, and it is formally verified. Simplex architecture by Sha et al. uses high assurance and high performance control subsystems [7]. The high assurance subsystem is used to keep the system within the safety envelope. ORTEGA architecture improves the Simplex architecture by adding on-demand detection and recovery of faulty tasks [8]. An anomaly based approach for detecting and identifying software and hardware faults in pervasive computing systems is proposed in [9]. The methodology uses an array of features to capture spatial and temporal variability to be used by an anomaly analysis engine.

Our work differs from these approaches by focusing on transient faults in a real-time system by using highly modular approach instead of redundancy. Similar solution based on modularity and fault isolation has been successfully used, e.g. in the non-real-time MINIX operating system (OS) for driver management [10].

3. Fault detection and recovery in real-time systems

3.1. System definition

Our hypothesis is that mission critical real-time systems can use service management to recover from transient faults (discussed in Section 5) in a loosely coupled software architecture. In this context, we define a loosely coupled real-time system as follows:

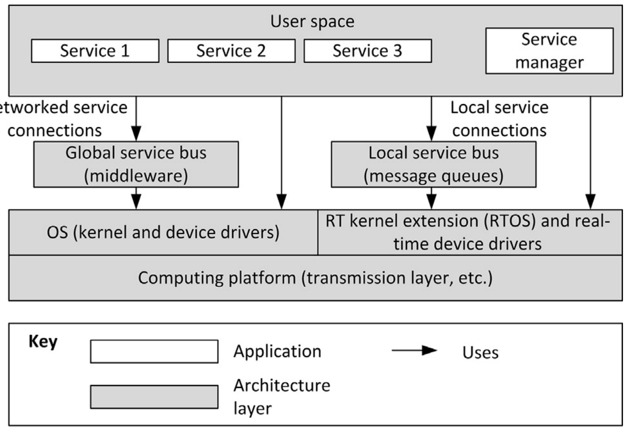
User space
Service 1
Service 2
Service 3
Service manager
Networked service connections
Local service connections
Global service bus (middleware)
Local service bus (message queues)
OS (kernel and device drivers)
RT kernel extension (RTOS) and real-time device drivers
Computing platform (transmission layer, etc.)
Key
Application
Architecture layer
Uses

Fig. 1. Logical system architecture.

- The system is made up of a set of periodic processes, i.e. services.
- Services are loosely coupled, having no direct interdependencies or references to each other.
- Services can be distributed over network or located on a single computer.
- Services communicate with a communication buses that facilitate monitoring of communication deadlines.

Advent of modern, powerful processors to RH systems provides a chance to mitigate the delays caused by the recovery process if the fault is detected before deadlines. In an optimal case, a fault can be detected and recovered before it causes service failures. If fault recovery causes exceeding of a deadline, other services can detect this and react accordingly by moving the system to a safe state while simultaneously isolating the fault. Since this recovery strategy does not rely on redundant versions, there is no need to maintain consistency between replicas, which is a major challenge for redundant systems [11]. We also leave formal methods out of the scope of our solution because of architectural limitations and costs associated with these methods are likely to be prohibitive for ITER.

3.2. Architecture

The system architecture in our implementation is based on RTSOA using data-centric middleware and an open source real-time operating system (RTOS). It provides decoupled connections for the services via local and global service buses (LSB and GSB) and includes a service manager to monitor and manage services (Fig. 1) [1].

The LSB is based on real-time queues, a communication mode provided by the RTOS. A message queue can be created by one service and used by multiple services that send and/or receive messages to the queue. GSB is a wrapper that uses Data Distribution Service for Real-Time Systems (DDS) middleware for networked connections; DDS is a standard for decentralized and data-centric middleware based on the publish/subscribe model and aimed at mission-critical and embedded systems. The standard is maintained by the Object Management Group (http://portals.omg.org/dds/).

3.3. Service manager and service configuration

Service manager is a local component used to start services and detect faults. Service manager spawns the services as child processes, according to a configuration file – more advanced configuration methods could be supported, e.g. GUIs, web interface