The need for fault isolation in object-oriented control systems

M. Larsson, I. Klein  
Dept. of Electrical Engineering  
Linköping University  
S-581 83 Linköping, Sweden  
magnusl, inger@isy.liu.se

D. Lawesson, U. Nilsson  
Dept. of Computer and Info. Science  
Linköping University  
S-581 83 Linköping, Sweden  
danla, ulfni@ida.liu.se

February 20, 1999

Report no.: LiTH-ISY-R-2098

Technical reports from the Automatic Control group in Linköping are available by anonymous ftp at the address ftp.control.isy.liu.se. This report is contained in the compressed postscript file 2098.ps.Z.
The need for fault isolation in object-oriented control systems

M. Larsson, I. Klein
Dept. of Electrical Engineering
Linköping University
S-581 83 Linköping, Sweden
magnus1, inger@isy.liu.se

D. Lawesson, U. Nilsson
Dept. of Computer and Info. Science
Linköping University
S-581 83 Linköping, Sweden
danla, ulfni@ida.liu.se

February 20, 1999

Abstract
This report discusses the problem with fault propagation in large scale control systems with object oriented architecture. There seems to be a trade-off between the degree of object encapsulation and the possibility of suppressing propagating error messages – when an individual object detects a fault, it does not in general know how close it is to the real fault, and hence whether it should report an error to the operator or not. Mechanisms for querying other objects on-the-fly is feasible only for closely related objects due to OO architecture goals.

Keywords: fault isolation, control system design, UML

1 Introduction
Developing control systems for complex systems is a difficult and increasingly important task. Larger control systems have traditionally been developed using structured analysis and functional decomposition (see e.g. DeMarco [1]). Using traditional programming, it is in principle possible for the complete system state to be known centrally, hence concise information to an operator about a fault situation can, at least in principle, be generated.

Today many large systems are designed using an object oriented approach (see e.g. [2, 3, 4]). This has several advantages over traditional approaches, including better possibility to cope with complexity and to achieve good maintainability and reusability (e.g. [5]). In object oriented design, encapsulation and modularity are fundamental and important design goals for reuse, maintenance and complexity reasons. It will be argued in this report how these object-oriented design goals often stand in direct conflict with the need to generate concise information about a fault situation.

2 Problem description
Our concern here is how a configurable and safety critical object oriented control system handles and isolates run-time faults and alarms, and specifically the
issues that arise due to the object oriented structure and complexity of the control system itself. In object oriented design, encapsulation and modularity are fundamental and important design goals for reuse, maintenance and complexity reasons. An often conflicting design goal lies in the need to suppress unnecessary, propagating, error messages and eventually give the operator a concise picture of a fault situation.

With the term fault, we will here mean a run-time change or event, normally in hardware, that eventually causes the system\(^1\) to abort normal operation. When a fault occurs during normal operation, the system often generates a large number of error messages. Error messages are sent by individual objects to notify an operator when the object has detected an error condition. The individual object does not in general know how close it is to the real fault or if sufficient reporting is already taking place, and hence whether it should report to the operator or not. For objects close to each other it is possible to suppress error messages by information passing, but this is not always feasible – it is an explicit aim of object oriented modeling to encapsulate knowledge about the internal state of objects and to achieve independence between groups of collaborating objects (i.e., encapsulation and modularity). Moreover, the control system that we consider here is safety critical. In case of a serious fault, the first priority is to take the system to a safe state, and only then is it possible to start analyzing what may have caused the fault.

Our primary concern is a situation where we have an operational system which is normally running without direct supervision. Moreover, it can be assumed that operators or service personnel summoned when the system halts are fairly unexperienced with the system and at least not normally has any insight in the internal design of the control system. Since the error messages stemming from a certain fault often reflects the control systems internal design and architecture, it can be very difficult for the operator to understand which error message that is most relevant and closest to the real fault.

### 3 System characteristics

The system we consider is an object oriented control system. As basic inspiration and as a case study we have used a control system for industrial robots developed by ABB Robotics. However, we believe that the system has many characteristic features in common with other control systems that are designed using the object oriented approach. In this section we will describe the main characteristics of the object oriented control system.

ABB Robotics has a family of industrial robots used for different purposes, see Figure 1. The control system is designed to handle all of these robots, which means that the control system is highly configurable depending on which robot a particular system is controlling, and what hardware devices that are used in the particular installation. The configuration is stored in a database for each control system. Furthermore, the user can program the system to perform different tasks. The control system is multi-threaded, and there are several concurrent tasks communicating both asynchronously and synchronously. The objects in the system are both pure software objects as well as objects corresponding to hardware. Many of the hardware devices will have a mirror object in the

---

\(^1\)With system we will mean the control system, if not otherwise clear from context.
control system, and one of the tasks for these mirror objects is to supervise the corresponding hardware and report when there is a problem.

Since we in this work concentrate on fault handling for a fully operational system, we will not discuss the installation and startup problems. To specify even more, we consider run-time faults that cause the system to abort normal operation. There are mainly two types of run-time faults that occur in the system: hardware faults and real-time faults. The real-time faults can be due to several reasons. Both synchronous and asynchronous interprocess communication may, e.g., be realized by a subscribe/notify pattern in which case the sender and receiver in principle are known to each other at run-time. Another case occurs when an object/task is an event consumer. In this case a fault is detected by an empty queue and who failed to send the expected data is often not known to the consumer.

We will give a short overview of how the control system handles faults internally. When an error condition is encountered, the current function normally returns with an error code. It might also decide to continue its normal operation, e.g., an event driven threads main loop. The returned error code in turn can be regarded as an error condition by the calling object. If deemed appropriate by the designers, the object registers an error message to a central log.

If an error condition is deemed so serious by a detecting object that normal operation cannot continue, a special asynchronous call is made that performs an emergency stop.
Exception handling

Exception handling mechanisms are intended to help improve error handling in software, to make programs more reliable and robust. They are language constructs that facilitate error handling outside of the normal program flow and at the appropriate level. The exception constructs might also support the programmer in providing more information to the error handler code than available through the normal object interface, to facilitate error recovery.

Exception handling mechanisms are to their nature low level constructs, and as such address the fault handling problem bottom up, while the scheme we propose in this article takes a more abstract view, and addresses the problem, mainly fault propagation, from above. The methods described in this paper are not intended to replace low level error handling, but to be used in conjunction with low level error handling in some form. It can e.g., be a disciplined use of return codes or full fledged exception handling mechanisms.

It is interesting to note that, as pointed out in [13], the goals of exception handling often stand in direct conflict with the goals of an object oriented architecture, the very same goals of encapsulation and modularity that cause the fault propagation problem addressed in this work.

References


