# Integrating COTS Software Components into Dependable Software Architectures

Paulo Asterio de C. Guerra
Cecília Mary F. Rubira
*Instituto de Computação*
*Universidade Estadual de Campinas, Brazil*

Alexander Romanovsky
*School of Computing Science*
*University of Newcastle upon Tyne, UK*

Rogério de Lemos
*Computing Laboratory*
*University of Kent at Canterbury, UK*

## Abstract

*This paper considers the problem of integrating commercial off-the-shelf (COTS) software components into systems with high dependability requirements. These components, by their very nature, are built to be reused as black boxes that cannot be modified. Instead, the system architect has to rely on techniques external with respect to the component for resolving mismatches of the services required and provided that might arise in the interaction of the component and its environment. This paper proposes an architectural solution to turning COTS components into idealised fault-tolerant COTS components by adding protective wrappers to them.*

## 1. Introduction

Current large-scale systems usually integrate COTS components that may act as service providers and/or service users. In order to build a dependable software system from untrustworthy COTS components, the system should treat these components as a potential source of faults. The overall software system should be able to support COTS components while preventing the propagation of errors.

In this paper we are mainly concerned with design faults that may exist at the application layer of a component-based system. We present the concept of an *idealised fault-tolerant COTS component*, which is an architectural solution that encapsulates a COTS component adding fault tolerance capabilities to allow it to be integrated in a larger system. These fault tolerant capabilities are related to the activities associated with error processing, that is, error detection and error recovery. The rest of the paper is organised as follows. In the next section, we discuss background work. Section 3 describes the architectural representation of idealised fault-tolerant COTS. The case study demonstrating the feasibility of the proposed approach is presented in section 4. Finally, section 5 presents some concluding remarks and discusses future work.

## 2. Background

An *architectural mismatch* occurs when the assumptions that a component makes about another component or the rest of the system do not match [4]. When building systems from existing components, it is inevitable that incompatibilities between the service delivered by the component and the service that the rest of the system expects from that component give rise to such mismatches. We view all incompatibilities between a COTS component and the rest of the system as architectural mismatches. This, for example, includes internal faults of a COTS component that affect other system components or its environment, in which case the failure assumptions of the component were wrong.

The C2 architectural style is a component-based style that supports large grain reuse and flexible system composition, emphasizing weak bindings between components [8]. Components communicate only through asynchronous messages mediated by connectors. Both components and connectors have a *top interface* and a *bottom interface*. Systems are composed in a layered style, where the top interface of a component may be connected to the bottom interface of a connector and its bottom interface may be connected to the top interface of another connector. Each side of a connector may be connected to any number of components or connectors. *Requests* from client components flow up through the system layers and responses from server components, called *notifications*, flow down.

## 3. Idealised Fault-Tolerant COTS Component

The idealised fault-tolerant COTS component is a specialization of the idealised C2 Component [5], which is briefly described in the following section.

## 3.1. The Idealised C2 Component

The idealised C2 component (iC2C) is equivalent, in terms of behaviour and structure, to the idealised fault-tolerant component [2] and designed to allow the structuring of software architectures compliant with the C2 architectural style [8]. Service requests and normal responses of an idealised fault-tolerant component are mapped as requests and notifications in the C2 architectural style. Interface and failure exceptions of an idealised fault-tolerant component are considered subtypes of notifications. The overall structure of the iC2C has two distinct components and three connectors, as shown in Figure 1.

The iC2C NormalActivity component implements the normal behaviour, and is responsible for error detection during normal operation, and the signalling of interface and internal exceptions. The iC2C AbnormalActivity component is responsible for error recovery, and the signalling of failure exceptions.

The iC2C connectors are specialized reusable C2 connectors with the following roles: (i) The iC2C_bottom connector connects the iC2C with the lower components of a C2 configuration, and serializes the requests received; (ii) The iC2C_internal connector controls message flow inside the iC2C, selecting the destination of each message received based on its originator, the message type and the operational state of the iC2C; and (iii) The iC2C_top connector connects the iC2C with the upper components of a C2 configuration.

## 3.2. COTS Component Protectors

A *wrapper*, in a broad sense, is a specialised component inserted between a component and its environment to deal with the flows of control and data going to and/or from the wrapped component [3]. Wrappers can be employed for improving quality properties of the components such as adding caching and buffering, dealing with mismatches or simplifying the component interface.

A systematic approach has been proposed for using protective wrappers, known as *protectors*, that can improve the overall system dependability [6]. This is achieved by protecting both the system against erroneous behaviour of a COTS component, and the COTS component against erroneous requests from the rest of the system. The approach consists of rigorous specification of the wrapper functionality in forms of *acceptable behaviour constraints* (ABCs) and in their execution at run time in the form of executable assertions [7] detecting a constraint violation and of exception handlers recovering after it. The general sources of information to be used in developing both ABCs and possible actions to be undertaken in response to their violations are the following:

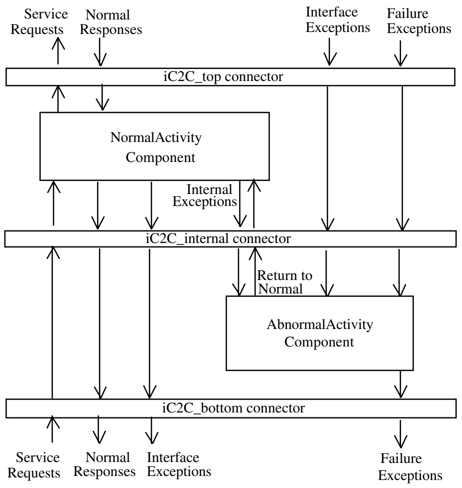

**Figure 1. Idealised C2 Component (iC2C)**

(i) The behaviour specification of COTS components as specified by the COTS's developers.

(ii) The behaviour specification of a COTS component as specified by the system designers. This description and the previous one must satisfy certain mutual constraints for the system design to be correct, but they will not be identical.

(iii) The behaviour that the system designer expects from a COTS component (not necessarily approving it), based on previous experiences with it, i.e., he/she may know that it often fails in response to certain legal stimuli.

(iv) Component (COTS or part of the rest of the system) behaviour that system designers considers especially unacceptable, without knowing whether it is likely or not.

(v) The behaviour specifications of the rest of the system.

ABCs are generally expressed as a set of assertions on the states of input and output parameters. In addition to that they can include assertions on the histories (sequences of requests and responses) that the protector has to collect, including temporal data, and assertions on the state of the system and its components, which are to be retrieved by the protector by calling side-effect-free functions.

## 3.3. Idealised C2 COTS (iCOTS)

A protective wrapper for a COTS software component is a special type of application-specific fault-tolerance capability. To be effective, the design of fault-tolerance capabilities must be concerned with architectural issues, such as process distribution and communication mode, that impact the overall system dependability. Although the C2 architectural style is specially suited for integrating COTS components into a larger system, its rules on topology and communication are not adequate for incorporating fault tolerance mechanisms into C2 software architectures, especially the mechanisms used for error detection and fault containment [5]. The idealised C2 fault-tolerant component (iC2C) architectural solution (section 3.1) overcomes these problems leveraging the C2 architectural style to allow such COTS software components to be integrated in dependable systems.

The idealised C2 COTS (iCOTS) is a specialization of the iC2C aiming to add protective wrappers to a COTS component to be integrated in a software system. The resulting failure behaviour of the idealised COTS is that of the original idealized fault-tolerant component. In our approach, the COTS component is encapsulated into the NormalActivity component of an iC2C, wrapped by two specialized connectors acting as error detectors (Figure 2). These detectors are responsible for verifying that the messages that flows to/from the COTS being wrapped do not violate the acceptable behaviour constraints (e.g., timing constraints) specified for that system. The lower_detector inspects incoming requests and outgoing responses (C2 notifications) from/to the COTS clients while the upper_detector inspects outgoing requests and incoming responses to/from other components providing services to the COTS.

When a constraint violation is detected, the detector sends an exception notification that will be handled by the AbnormalActivity component, following the rules defined for the iC2C. Any of these detectors may be decomposed in a set of special purpose error detectors which, in their turn, are wrapped by a pair of connectors. For example, Figure 3 shows an upper_detector decomposed into a number of error detectors. The detector_bottom coordinates error detection, and the detector_top connects the whole detector either to the COTS or to the iC2C top_connector. The AbnormalActivity component is responsible for both error diagnosis and error recovery. Depending on the complexity of these tasks, it may be convenient to decompose it into more specialized components for error diagnosis and an error handling.

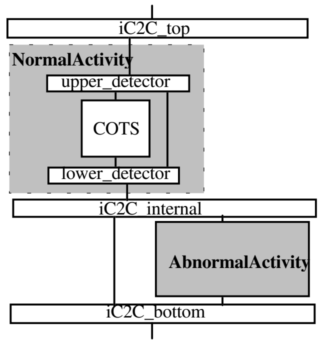

**Figure 2. Idealised C2 COTS Overall Structure**

## 4. Case Study

Anderson et. al. [1] present the results of a real-time case study in protective wrapper development, in which a Simulink model of a steam boiler system is used together with an off-the-shelf PID (Proportional, Integral and Derivative) controller. The protective wrappers are developed to allow detection of and recovery from typical errors caused by unavailability of signals, violations of limitations, and oscillations. In this section, we describe an architectural solution for the steam boiler system conforming to the C2 architectural style and applying the idealised C2 COTS (iCOTS) previously described (section 3.3).

The proposed C2 architectural configuration is organized in four layers: (i) the BoilerController component; (ii) the WaterFlowController and CoalFeederController; (iii) the AirFlowController, which has as input the CoalFeederRate from the CoalFeederController; and (iv) the sensors and actuators required by the system.

In the following we describe how an iCOTS AirFlowController can be built following our general proposal, encapsulating a COTS PID controller with protectors. The same approach can be applied to the WaterFlowController and CoalFeederController.

Figure 4 shows the internal structure of the iCOTS for the AirFlowController, based on Figure 2. The COTS PID controller is wrapped by a pair of error detectors (upper_detector and lower_detector) and inserted into an iC2C as its NormalActivity component. Both detectors use OscillatorChecker, which is responsible for checking whether oscillating variables revert to a stable state before a maximum number of oscillations. Table 1 specifies, for the upper_detector, the message types to be inspected, their corresponding assertions that guarantee the acceptable behaviour constraints (section 3.2) and the type of the exception notification that should be generated when a constraint is violated.

In this example, the recovery actions are delegated to the BoilerController component, which may either sound an alarm or shut down the system, depending on the exception type. Thus, internal exceptions are propagated by the AFCErrorHandler as failure exceptions of a generic type. An InvalidAirFlowRate exception, for example, will

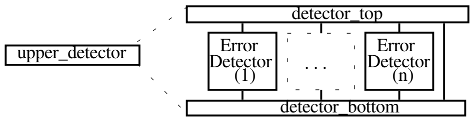

**Figure 3. Decomposition of a Detector**

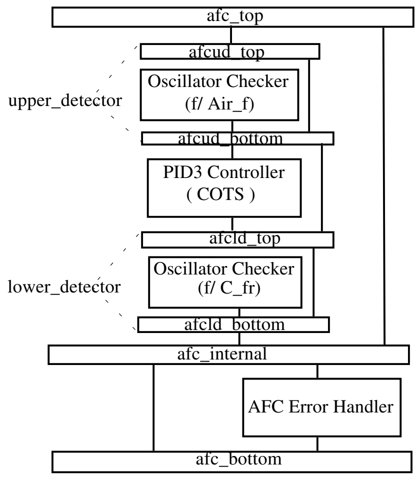

**Figure 4. Decomposition of the AirFlowController**

generate an OutOfRange failure exception.

## 5. Conclusions and Future Work

The paper proposes an architectural solution to turning COTS components into idealised fault-tolerant COTS components by adding protective wrappers to them. We demonstrate the feasibility of the proposed approach using the steam boiler system case study, where its controllers are built reusing unreliable COTS components. The costs associated with our approach in terms of increasing the complexity of the design and run-time overhead are dependent on (i) the granularity of the COTS components to be protected and (ii) their coupling with the rest of the system. The ideal scenario to minimize those costs is to choose large-grained COTS components which are integrated to the system through a restricted set of well-defined interfaces. In this scenario, the size and complexity of the protectors will be proportionally small relative to the COTS granularity. So, the tradeoffs between those costs and the benefits resulting from the improvement in reliability can easily justify the applicability of our approach to real-world complex systems.
Although the case study has used a single architectural style, software components in the C2 architectural style can be integrated into configurations of other architectural styles using simple adapters. This allows the idealised fault-tolerant COTS (iCOTS) concept to be applied as a general solution in developing dependable systems from unreliable COTS components. Our future work includes extending an existing C2 Java framework [9] to aid in the implementation of this new abstraction and its integration into C2 architectural configurations. Another future direction is to apply the proposed approach to larger real-world systems; in order to verify if it scales as well as the structuring concepts from what it was derived.

## Acknowledgments

P. Guerra is supported by CAPES/Brazil. C. Rubira and P. Guerra are supported by the FINEP/Brazil Project PRONEX-SAI-7697102200. C. Rubira is also supported by CNPq/Brazil (351592/97-0). A. Romanovsky is supported by IST DSoS and EPSRC/UK DOTS Projects.

## References

[1] T. Anderson, M. Feng, S. Riddle, A. Romanovsky. Protective Wrapper Development: A Case Study. In *Proc. 2nd Int. Conference on COTS-based Software Systems*. Ottawa, Canada. Feb., 2003.

[2] T. Anderson, P. A. Lee. *Fault Tolerance: Principles and Practice.* Prentice-Hall, 1981.

[3] R. DeLine. "A Catalog of Techniques for Resolving Packaging Mismatch". In *Proc. 5th Symposium on Software Reusability (SSR'99).* Los Angeles, CA. May 1999. pp. 44-53.

[4] D. Garlan, R. Allen, J. Ockerbloom. Architectural mismatch: Why reuse is so hard. *IEEE Software*, 12(6):17--26, Nov. 1995.

[5] P. A. C. Guerra, C. M. F. Rubira, R. de Lemos. An Idealized Fault-Tolerant Architectural Component, In *Proc. ICSE 2002 Workshop on Architecting Dependable Systems*, Orlando, USA, 2002, pp. 15--20.

[6] P. Popov, S. Riddle, A. Romanovsky, L. Strigini. On Systematic Design of Protectors for Employing OTS Items. In *Proc. 27th Euromicro conference*. Warsaw, Poland, 4-6 Sep., IEEE CS, 2001. pp.22-29.

[7] D. S. Rosenblum. Towards a Method of Programming with Assertions. In *Proc. 14th International Conference on Software Engineering*. 1992.

[8] R. N. Taylor, et.al. A component- and message-based architectural style for GUI software. *IEEE Transactions on Software Engineering*, 22(6):390--406, June 1996.

[9] UCI. Archstudio 3 - Foundations - c2.fw, http://www.isr.uci.edu/projects/archstudio/c2fw.html.

**Table 1. Error Detection Specifications**

| **Message Type** | **Constraints to be checked** ***Exceptional Notification*** |
|---|---|
| Request setAirFlow(Air_f) | 0 <= Air_f <= 0.1 *InvalidAirFlowRate* |
| | check_oscillate(Air_f) AirFlowRateOscillating |
| | the corresponding notification must be received within a specified time interval *AirFlowActuatorTimeout* |
| Notification from readO2Concentration() | 0 <= O2eco <=1 *InvalidO2Concentration* |