# A Fault-Tolerant Software Architecture for COTS-Based Software Systems

Paulo Asterio de C. Guerra
Cecília Mary F. Rubira
Instituto de Computação
Universidade Estadual de Campinas
C.P. 6156 Campinas, SP - 13083-970
Brazil
+55 (19) 3788-5839
{asterio,cmrubira}ic.unicamp.br

Alexander Romanovsky
School of Computing Science
University of Newcastle upon Tyne
Newcastle upon Tyne, NE1 7RU, UK
+44 (191) 222-8135
alexander.romanovsky@ncl.ac.uk

Rogério de Lemos
Computing Laboratory
University of Kent at Canterbury
Canterbury, Kent, CT2 7NF, UK
+44 (1227) 823628
r.delemos@kent.ac.uk

## ABSTRACT
This paper considers the problem of integrating Commercial off-the-shelf (COTS) components into systems with high dependability requirements. Such components are built to be reused as black boxes that cannot be modified. The system architect has to rely on techniques that are external to the component for resolving mismatches between the services required and provided that might arise in the interaction of the component and its environment. The paper puts forward an approach that employs the layer-based C2 architectural style for structuring error detection and recovery mechanisms to be added to the component during system integration.

## Categories and Subject Descriptors
D.2.11 [**SOFTWARE ENGINEERING**]: Software Architectures – *domain-specific architectures, patterns.*

## General Terms
Design, Reliability, Security.

## Keywords
Software architecture, COTS-based, Fault-tolerance.

## 1. INTRODUCTION
A commercial off-the-shelf (COTS) component is usually provided as a black box to be reused "as it is". These components usually do not have complete rigorously written specification, there is no guarantee that the description the integrators have in their disposal is correct (very often it is ambiguous). These components can have bugs, moreover, the specific context in which they are used is not known at their development time. Furthermore, it may be imperative to replace a COTS component already integrated in a stable system by a new version released without the control of the system integrator and that may contain new bugs. When integrating such a component into a system with high dependability requirements we should employ solutions at the architectural level to ensure that these requirements are met, irrespective of faults in the COTS component itself or in the way it interacts with the other system components.


*ESEC/FSE'03*, September 1–5, 2003, Helsinki, Finland


Research into describing software architectures with respect to their dependability properties has recently gained considerable attention [10, 11]. In [4] the idealised fault-tolerant component concept [1] is applied in the architectural description of fault-tolerant component-based systems. [9] puts forward a general approach to developing protective wrappers to be used for building dependable software systems based on COTS components. In this paper we combine the concepts of an idealised architectural component and protective wrappers to develop an architectural solution that provides an effective and systematic way for building dependable software systems from COTS software components.

The rest of the paper is organised as follows. In the next section, we briefly discuss background work on the idealised fault-tolerant component, architectural mismatches and the C2 architectural style. Section 3 describes the architectural representation of idealised fault-tolerant COTS. Finally, section 4 presents some concluding remarks and discusses our future work.

## 2. BACKGROUND

### 2.1. The Idealised Fault-Tolerant Component
The idealised fault-tolerant component is a structuring concept for the coherent provision of fault tolerance in a system. Through this concept, we can allocate fault-tolerance responsibilities to the various parts of a system in an orderly fashion, and model the system recursively, such that each: component can itself be considered as a system on its own, which has an internal design containing further sub-components [1]. The communication between idealized fault-tolerant components is only through request/response messages. Upon receiving a request for a service, an idealised component will react with a normal response if the request is successfully processed or an external exception, otherwise. This external exception may be due to an invalid service request, in which case it is called an interface exception, or due to a failure in processing a valid request, in which case it is called a failure exception. Internal exceptions are associated with errors detected within a component that may be corrected,

allowing the operation to be completed successfully; otherwise, they are propagated as external exceptions. An idealized component must provide appropriate handlers for all exceptions it may be exposed to. Thus, the internal structure of an idealized component has two distinct parts: one that implements its normal behaviour, when no exceptions occur, and another that implements its abnormal behaviour, which deals with the exceptional conditions.

## 2.2. Architectural Mismatches and COTS Component Integration

Architectural mismatch [3] is one of the most difficult problems faced by system integrators when building systems from COTS components. An *architectural mismatch* occurs when the assumptions that a component makes about another component do not match. That is, the assumptions associated with the service provided by the component are different from the assumptions associated with the services required by the component for behaving as specified [8]. 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. These mismatches are not exclusive to the functional attributes of the component; mismatches may also include quality attributes, such as dependability, which can be related to the component failure mode assumptions or its safety integrity levels.

We view all incompatibilities between components of a system as architectural mismatches. This, for example, includes internal faults of a COTS component that affect others system components or its environment, in which case the failure assumptions of the component were wrong.

## 2.3. The C2 Architectural Style

The C2 architectural style is a component-based style that supports large grain reuse and flexible system composition, emphasizing weak bindings between components [12]. Both components and connectors in C2 have a *top interface* and a *bottom interface*. Systems are composed in a layered style, where the top (bottom) interface of a component may be connected to the bottom (top) interface of a connector. Each side of a connector may be connected to any number of components or connectors. Components communicate only through asynchronous *requests* and *notifications* messages that are filtered and broadcasted by the connectors. By convention, *requests* flow up through the system layers and *notifications* flow down. In response to a request, a component may emit a notification back to the components below. Upon receiving a notification, a component may react with the *implicit invocation* of one of its operations.

# 3. Idealised Fault-Tolerant COTS Component

Current large-scale systems usually integrate COTS components that may act as service providers and/or service users. Since, there is no control, or even full knowledge, over the design, implementation and evolution of COTS components, the evolutionary process of a COTS component should be considered as part of a complex environment, physical and logical, that might directly affect the system components. 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 other words, the system should be able to tolerate faults that may reside or occur inside the COTS components, while not being able to directly inspect or modify its internal state or behaviour.

In this paper, 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 idealised fault-tolerant COTS component is a specialization of the idealised C2 Component (iC2C) [4], which is briefly described in the following section.

## 3.1. The Idealised C2 Component (iC2C)

In terms of behaviour and structure the idealised C2 component (iC2C) is equivalent to the idealised fault-tolerant component [1] and is designed to allow software architectures to be structured to be compliant with the C2 architectural style [12]. Service requests and normal responses of an idealised fault-tolerant component are represented as requests and notifications in the C2 architectural style. Interface and failure exceptions of an idealised fault-tolerant component are mapped into subtypes of notifications. In order to minimize the impact of fault tolerance provision on the overall system complexity, we decouple the normal activity and abnormal activity parts of the idealised component. These decisions lead us to an overall structure for the iC2C that has two distinct components and three connectors, as shown in Figure 1.

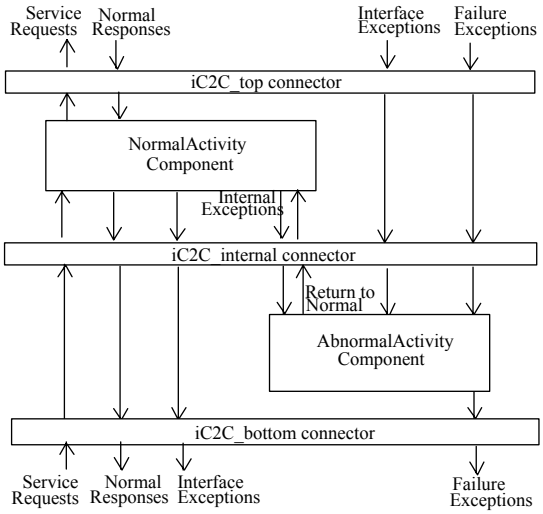


**Figure 1. Idealised C2 Component (iC2C)**

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

Component wrapping is a well-known structuring technique that has been used in several areas. In this paper we use term "wrapper" in a very broad sense, incorporating the concepts of wrappers, mediators, and bridges (as referred in [2]). A *wrapper* is a specialised component inserted between a component and its environment to control the flows of control and data going to and/or from the wrapped component. The need for wrapping arises when (i) it is impossible or expensive to change the components when reusing them as parts of a new system, or (ii) if it is easier to add new features by incorporating them into wrappers (which is typically the case for the COTS items). Wrapping is a structured and a cost-effective solution to many problems in component-based software development. 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. With respect to dependability, wrappers are usually used for ensuring security, transparent component replication, etc.

Paper [9] puts forward a systematic approach to using protective wrappers, called *protectors*, which can improve the overall system dependability. 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 wrappers are viewed as redundant software that detects errors or suspicious activity and executes appropriate recovery when possible.

The development of protectors is considered to be a part of system integration activities [9]. The approach consists of rigorous specification of the wrapper functionality in forms of *acceptable behaviour constraints* (ABCs) that are verified at run time to detect errors and enable a forward error recovery strategy based on exception handling. 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:

(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. E.g., the system designer's description requires the COTS component to be able to react to a set of stimuli that is a subset of the set specified by the COTS's developers.

(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) COTS or system behaviour that designers consider especially unacceptable, without knowing whether it is likely or not.

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

The sources of information above allow the developer to formulate a number of statements describing the correct behaviour of the COTS component and which it expects from the rest of the system. The statements are expressed as a set of executable assertions on the states of input and output parameters. In addition to that they can include assertions on the histories (sequences of calls) that the protector has to collect and assertions on the states of the system components that are to be retrieved by the protector by calling side-effect-free functions returning the states of these components. While [9] deals with the design of COTS protectors and its development process, in this paper we are mainly concerned with architectural issues related to their integration in a fault-tolerant component-based system.

## 3.3. Idealised C2 COTS (iCOTS)

A protective wrapper for a COTS software component is a specialized software developing application-specific fault-tolerance. 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 [4]. 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. In our approach, the COTS component is encapsulated into the NormalActivity component of an iC2C, wrapped by two specialized connectors acting as error detectors (see Figure 2).

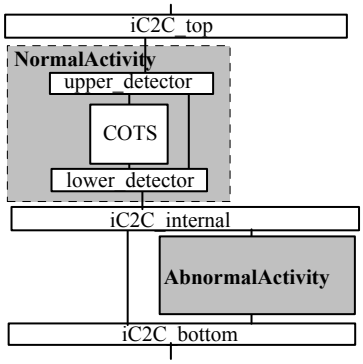


**Figure 2. Idealised C2 COTS (iCOTS) Overall Structure**

The main responsibility of these detectors is to verify that the messages that flow to/from the wrapped COTS do not violate the acceptable behaviour constraints specified for the system. The lower_detector inspects both 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 surrounding components providing services to the COTS component.

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. The AbnormalActivity component may, for instance, decide to adjust parameters of an invalid service request to allow its safe completion by the COTS component.

## 4. Conclusions and Future Work

When building reliable systems from existing components, guarantees cannot be given on the system behaviour, if no guarantees are provided on the behaviour of its individual components. Since no such guarantees are provided for individual COTS components, architectural means at the component level have to be devised for the provision of the necessary guarantees at the system level.. The paper proposes an architectural solution to turning COTS components into idealised fault-tolerant COTS components by adding protective wrappers to them. Generally speaking, there are two ways of dealing with mismatches: mismatch removal and mismatch tolerance [6]. Our architectural solution can be used to employ both of these approaches. A small case study demonstrating the feasibility of the proposed approach can be found in [5]. We are now working on the implementation of a real-world application employing the approach here proposed for interacting with a COTS Geographic Information Systems. This work has the support of a Java framework [7] that aids in the implementation of the iC2C abstraction and its integration into C2 architectural configurations.

Although we recognize that the proposed approach can result in incorporating repetitive checks into the integrated system, this is an unavoidable outcome considering the lack of guarantees provided by COTS components. For example, it might be the case that a COTS component has internal assertions checking the validity of an input parameter that is also checked by its protector, or other protectors developed for other COTS components. However, there are situations in which the system integrator can take care of this by coordinating development of fault tolerance means associated with individual components - this is one of possible directions of further development of our approach. Our future work will be centred on the smooth integration of the protector design into the development process. This includes architectural level specification of abstract ABCs to be refined and/or extended during the design and implementation phases, possibly allowing automatic generation of the iCOTS from these specifications. Another direction of our future work, to be done after the proposed solution has been applied to a number of real-word software systems, is to define a new architectural pattern based on it.

The solution described in the paper is based on the C2 architectural style, but it can be easily adapted to other more common message-based styles, such as the broker style employed by CORBA. Also, configurations using the C2 style can be integrated into heterogeneous style architectures by means of 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.

## Acknowledgments

P. Guerra is supported by CAPES/Brazil. C. Rubira is supported by CNPq/Brazil (grant 351592/97-0). A. Romanovsky is supported by IST DSoS and EPSRC/UK DOTS Projects.

## References

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

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

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

[4] Guerra, P. A. C., Rubira, C. M. F., de Lemos, R. A Fault-Tolerant Architecture for Component-Based Software Systems. In R. de Lemos, C. Gacek, A. Romanovsky (Eds). *Architecting Dependable Systems*. LNCS 2677. Springer Verlag. pp. 175-194. 2003.

[5] Guerra, P. A. C., Rubira, C. M. F., Romanovsky, A. and de Lemos, R. Integrating COTS Software Componentes into Dependable Software Architectures. In *Proc. 6th IEEE ISORC'2003*, Hokkaido, Japan, 2003, pp. 139-142.

[6] Lemos, R., Gacek, C., Romanovsky, A. Tolerating Architectural Mismatches. In R. de Lemos, C. Gacek, A. Romanovsky (Eds). *Architecting Dependable Systems*. LNCS 2677. Springer Verlag. pp. 175-194. 2003.

[7] Lima Filho, F.J.C., Guerra, P.A.C., Rubira, C.M.F. *FaTC2: An Object-Oriented Framework for Developing Fault-Tolerant Component-Based Systems*, In *Proc. ICSE 2003 Workshop on Architecting Dependable Systems*, Portland, USA, 2002, pp. 13--18.

[8] Oberndorf, P., Wallnau, K., Zaremski, A. M. Product Lines: Reusing Architectural Assets within an Organisation. *Software Architecture in Practice*. Eds. L. Bass, P. Clements, R. Kazman. Addison-Wesley. 1998. pp. 331-344.

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

[11] Saridakis,T. , Issarny, V. Developing Dependable Systems using Software Architecture. In *Proc. 1st Working IFIP Conf. on Software Architecture*, pp. 83-104, February 1999.

[10] Sotirovski, D. Towards Fault-Tolerant Software Architectures. In *Working IEEE/IFIP Conference on Software Architecture*. Eds. R. Kazman, P. Kruchten, C. Verhoef, H. Van Vliet, pp. 7-13, Los Alamitos, CA, USA, 2001.

[12] Taylor, R.N., Medvidovic, N., Anderson, K.M., Whitehead Jr., E.J., Robbins, J.E., Nies, K.A., Oreizy, P. and Dubrow, D.L. A Component- and Message-based Architectural Style for GUI Software. *IEEE Transactions on Software Engineering*, 22(6):390--406, June 1996.