School of Computing Science,
University of Newcastle upon Tyne

Protective Wrapping of Off-the-Shelf Components

Meine van der Meulen, Steve Riddle, Lorenzo Strigini, and Nigel Jefferson

Technical Report Series

CS-TR-857

August 2004

Protective Wrapping of Off-the-Shelf Components

Meine van der Meulen[1], Steve Riddle[2], Lorenzo Strigini[1], and Nigel Jefferson[2]

[1] Centre for Software Reliability, City University, London, U.K.
WWW home page: http://www.csr.city.ac.uk
[2] School of Computing Science, University of Newcastle upon Tyne, U.K.
WWW home page: http://www.csr.ncl.ac.uk

Abstract. System designers using off-the-shelf components (OTSCs), whose internals they cannot change, often use add-on "wrappers" to adapt the OTSCs' behaviour as required. In most cases, wrappers are used to change "functional" properties of the components they wrap. In this paper we discuss instead "protective wrapping", the use of wrappers to improve the dependability – i.e., "non-functional" properties like availability, reliability, security, and/or safety – of a component and thus of a system. Wrappers can improve dependability by adding fault tolerance, e.g. graceful degradation, or error recovery mechanisms. We discuss the rational specification of such protective wrappers in view of system dependability requirements, and highlight some of the design trade-offs and uncertainties affecting system design with OTSCs and wrappers, and differentiating it from other forms of fault-tolerant design.

1 Introduction

As building "component-based" software systems becomes more common, it becomes more often necessary to combine existing components – hardware as well as software – that were not necessarily designed to work together. *Wrapping* is a popular, often cost-effective technique for integrating pre-existing components into a system. When designing a new system, ad hoc "wrappers" are developed, i.e. new, small components that will be interposed between the others, reading and altering the contents of the communications they exchange. Wrapping has the advantage of not requiring detailed knowledge of the internal structure of the components being wrapped.

In most cases, wrappers are used to adapt the functionality of a component to the requirements set for it by the system's design: they often perform simple functions like translation between the argument formats used by two communicating components. In this paper we look instead at the use of wrappers for improving dependability. We call such wrappers *protective* wrappers. Protective wrapping is a way of structuring the provision of standard fault tolerance functions, like error detection, confinement and recovery, plus the less common function of *preventing* component failures, in a component-based design where dependability is a concern. We wish to clarify how these wrappers can be rationally specified, the trade-offs facing system designers (simply "designers" for

the rest of the paper), and the peculiarities of this form of fault-tolerant design, compared to the general case.

When designing a system with off-the-shelf components ("OTSCs"), it is often the case that an OTSC's functionality, and even more often its dependability, is insufficiently documented. Both these deficiencies are threats to system dependability: wrong assumptions about how an OTSC is intended to behave lead to system design faults; optimistic assumptions about an OTSC's probability of behaving as intended may lead to overestimating the dependability levels achieved by the chosen system design. Wrapping can help a designer to compensate for this lack of information.

Wrapping for dependability has been addressed by other authors. Wrappers are used to transform or filter unwanted communications that may cause failures. Fault injection may be used to identify such failure-causing values [7,3,5]. Wrappers are proposed to protect OTS applications that do not deal properly with kernel-raised exceptions, by transforming these into other exceptions or error return codes [7]; or to protect OTS kernels against inappropriate requests ([3]; here, an extended notion of wrappers is proposed that can access the kernel's internal data). In [5], the goal is automatic protection of library components against failure-causing parameter values, submitted by accident or malice. In [4], wrappers protect name servers from receiving unverifiable requests. A somewhat general approach to wrappers for common security concerns is described in [6].

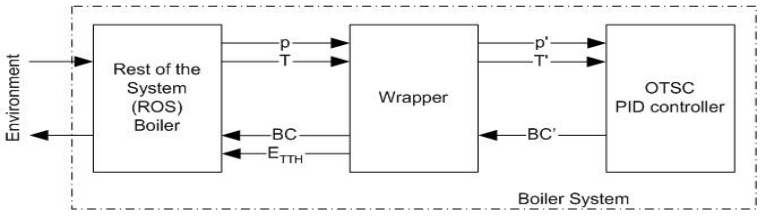
Most of this previous work assumes that a good knowledge can be gained about which communications will cause OTSC failure. We have argued for a more general view of protective wrapping [9], to take into account the fact that this knowledge is usually deficient, the specification of the OTSC may be incomplete, and designers need to be concerned with failures of both the OTSC and the ROS. Here, we discuss issues of design, verification and quantitative dependability trade-offs that arise in protective wrapping.

In the rest of this paper, Section 2 introduces terminology and an illustrative example. Section 3 introduces the specifications of components in relation to system-level requirements, including those concerning fault tolerance. Sections 4 and 5 discuss the options for the actual semantics of wrappers, i.e. the "cues" that can trigger their intervention and the forms of these interventions. Section 6 sets the previous discussion of wrapper specifications in the context of probabilistic system dependability requirements and discusses the important design trade-offs that arise. Our conclusions follow.

2 System Model and Example

Throughout this paper, we will use a simple example to clarify the concepts introduced. The example system (Figure 1) is a water boiler. We focus on a single OTSC, in this case a PID (Proportional-Integral-Derivative) controller which provides feed-back control for the burner of the boiler, and on its communications with the rest of the system, seen as a single black box ("ROS"); the ROS may contain other OTSCs. This example omits some of the possible complica-

Fig. 1. The boiler control system used as an example.



tions of a real system (an OTSC may have direct communication links with the environment around the system, or communications with the ROS that cannot be intercepted by a wrapper) but will suffice for this brief discussion. We do not make any assumptions about whether the OTSC, ROS and wrapper are realised in hardware or software (or both).

The ROS outputs readings (p, T) of pressure and temperature in the boiler, and accepts a burner control input, BC, and an exception signal, E_{TTH}, which causes an alarm to a human operator. The OTSC accepts as inputs two real numbers (p', T') and a *reset* signal, and outputs a (real-valued) control signal for the burner, BC'.

The designer is concerned with the dependability of this system: how frequently the components will behave "abnormally" (will fail), whether these component failures will cause system failure, and whether the frequency and severity of these failures will be acceptably low. Because of this concern, instead of connecting the ROS outputs directly to the OTSC's inputs and vice versa, the designer introduces a protective wrapper between the ROS and the OTSC, as depicted, which transforms p into p', etc.

The wrapper monitors communications between the ROS and OTSC, and possibly changes the values transmitted to the ROS or the OTSC. The ROS sees the combination of the OTSC and wrapper as one component, which we call the "wrapped OTSC" (WOTSC); likewise, the OTSC sees a "wrapped ROS".

For the sake of simplicity, we assume here that the OTS and ROS, if connected without the protective wrapper, would, in the absence of failures, produce the combined behaviour required from the system. So, the OTSC in Figure 1 does not need "functional" wrapping, limiting our discussion to "protective" wrapping.

3 Roles of Components and Protective Wrappers

3.1 System requirements, components and interfaces

The designer's problem is how to ensure the required behaviour of the whole system, using a given OTSC. When considering dependability, a designer usually deals with multiple sets of requirements on system behaviour. First, there is a specified "nominal behaviour": what the system ought to do, at least if none of

its components fail. The designer usually has an understanding of a "nominal behaviour" for each component, and makes sure that if all components exhibit their nominal behaviours, then so will the system. Making the system fault-tolerant means ensuring that even if components violate their nominal behaviours (they fail), the system will still exhibit nominal behaviour (failure masking) or some "degraded" but acceptable behaviour (graceful degradation) or at least will remain within an envelope of "safe behaviours"; the choice being determined by the system dependability requirements and by the costs of these various options.

The complete dependability requirements will inevitably be probabilistic: in addition to defining a nominal behaviour and zero or more degraded behaviours or "modes" of operation, it will include required probabilities of these assertions holding during operation of the system[1]. A similar hierarchy of a nominal behaviour and more or less acceptable failure behaviours applies to dependability requirements for any component or subsystem.

In this and the next two sections, we will discuss the deterministic part of these dependability properties. In a proper design, the specified system-level properties need to be *verifiable*, in the sense that, given clear descriptions of how the various components will behave (in their nominal and degraded modes) and of their connections, one can deduce that the requirements for the whole system (for a nominal or degraded mode, as specified) are satisfied. The expected or required behaviours (*models* and *specifications* in what follows) of the components and of the system need to be described in some unambiguous language, e.g., preconditions and postconditions characterising the relation between sequences of their inputs and outputs [8].

These descriptions need not specify all details of behaviour of a component, i.e. they may be partial specifications. We might for instance describe a component in a numerical library as computing a certain floating-point result with a relative error of less than 1%, although in reality the relative error is smaller, and variable; or, rather than trying to describe in detail what a component would do if it failed, we would rather describe an envelope of plausible behaviours it may exhibit, and prove that some system-level requirement will be satisfied provided the component remains within that envelope.

The behaviour that the designer expects the OTSC, as procured, to exhibit can be described abstractly as pairs of pre and post-conditions [8]. The looser the postconditions (the fewer the restrictions assumed on the behaviour of the OTSC), the more arbitrary behaviours of the OTSC one will need to require the wrapper and ROS to cope with in order to guarantee any given system-level requirement. This may make the system more robust, but at a cost, which will be the more acceptable, the more likely the extra erroneous behaviours allowed by

[1] It is true that such a formal way of specifying dependability requirements is only in common use for few categories of systems. For many everyday systems, probabilities may not be mentioned at all, for instance. Yet, we think that any rational definition of requirements will include some idea of what probabilities would be unacceptably high for each given failure (i.e., degraded behaviour) mode, and a partial ordering between more and less acceptable modes.

the less restrictive model of the OTSC are in reality. Symmetrical considerations apply to the designer's expectations about the ROS's behaviour.

3.2 The Models of the OTSC and ROS

We assume that the designer has chosen a particular OTSC, either procured on the market or already available within the same company. For an OTSC from the commercial market, the documentation will often be of lower quality and procuring extra information is often cumbersome and expensive; on the other hand, if the component is in frequent use, the supplier may have reliable data on its dependability. Publicly available dependability data – e.g., collections of bug reports for software packages – may also be valuable, if they exist. Relevant other information about the OTSC may concern maintenance requirements, failure modes and their failure rates.

The documentation of the OTSC may not specify its behaviour in certain circumstances, and the designer's most prudent approach would then be to assume that it is completely undetermined. At the opposite extreme, designers may choose to guess the OTSC's behaviour, based on previous experience, expert knowledge or other information.

By contrast, the designer may have a more precise model of the ROS, if designed ad hoc or if it also uses wrapping to ensure predictable behaviour.

Boiler example A specific PID controller has been chosen as the OTSC. Suppose that its documentation is unclear about what happens when either p or T is negative. The designer's model of the OTSC may then prudently assume its behaviour as undefined when these preconditions are violated. There may be other preconditions, documented or suspected, for the PID controller to behave properly, e.g., upper bounds on the values and rates of change of p' and T'.

As for the model of the ROS, to prove that the system has correct (nominal) behaviour if no component fails, the designer will use a model that includes the sensors and actuators, the physical properties of the burner, the fluid in the boiler, etc. This alone may not guarantee the above preconditions for nominal behaviour of the OTSC. It will then be the wrapper's task to guarantee them.

3.3 Requirements on the Wrapped OTSC and ROS

The designer's specification for the WOTSC may differ from the model of the OTSC even in its nominal behaviour, e.g. by hiding from the ROS some of the functions offered by the OTSC. In addition, the WOTSC specification has to describe dependability requirements, which determine the need for fault tolerance provisions in the wrapper.

Boiler example The boiler needs from the PID controller a control signal, BC, derived from the pressure and the temperature of the boiler according to a PID control law. A degraded, safe behaviour from the system viewpoint is to switch off the boiler ($BC = 0$). Knowing that the OTSC's behaviour is undefined for negative p' or T', the designer may then specify that the WOTSC must behave like the OTSC, if $p \geq 0$ and $T \geq 0$, but if not, it must set BC to 0.

In addition, since the precondition for nominal behaviour of the OTSC requires $p' \geq 0$ and $T' \geq 0$, the designer may specify that the WROS must guarantee these properties (e.g. if $p < 0$, p' will be 0), All these specifications together define the specifications of the wrapper. Since the wrapper alters the interface behaviour of the ROS and OTSC, the designer needs to verify that these modified behaviours imply the required system behaviour. For instance, at the interface of the ROS with the wrapper, the ROS sees a component that behaves (nominally) as a PID controller except that, if p or T is negative, its inputs and output are "clamped" to zero.

4 Specifying the Protective Wrapper: Cues for Intervention

Usually, designers of fault-tolerant systems use the detection of errors to trigger defensive actions. This depends on a fairly accurate knowledge of the behaviour of all components when failure-free. In designing with OTSCs, though, this knowledge cannot be assumed. Furthermore, the design of OTSCs often precludes close monitoring for early error detection. So, designers may want their wrappers to react to a pattern of component behaviour that merely suggests a failure, although it may be correct, especially if the type and circumstances of the suspected failure would cause severe consequences to the system.

So, designers may take an attitude similar to that frequently taken in designing for safety: aiming more at keeping the behaviour of components within an envelope of behaviours that prevent unacceptable damage at system level, than at guaranteeing their correct (nominal) behaviour. They also face the same kind of trade-offs: the interventions of the wrapper will usually prevent some requested operation of the OTSC, possibly providing in its place a safe failure, or an alternative, degraded or less efficient service. Designers thus know that the more cues they decide to react to, the less likely the system will be to fail in unpredictable ways, but also the more likely for wrapper interventions to be the result of false alarms, and the more degradation in performance or availability.

The wrapper, as depicted in Figure 1, monitors the outputs of the ROS and of the OTSC for cues, and can manipulate their values before forwarding them to the corresponding inputs of the OTSC and of the ROS, respectively. It can also insert communications not initiated by the ROS or OTS, for instance exception signals in response to cues it has detected.

In the wrapper's specifications, pre-conditions about the possible cues will be matched with postconditions about actions for the wrapper to take in response.

5 Examples of Specifications for Wrapper Actions

For any given cue, the designer may choose among various possible reactions by the wrapper, depending on the system's architecture and dependability requirements. A few possible reactions were described in Sect. 3. We now discuss other

possibilities for providing fault tolerance via the wrapper. Some of these have been applied in our project in a case study in a simulated environment [1].

For instance, let us consider the case in which the ROS fails and issues a "suspicious" p value, e.g. a negative value, violating a precondition for the PID controller, whose behaviour is then unspecified. As in Sect. 3, the wrapper could mitigate the consequences of such a failure by *substituting this erroneous, dangerous or suspicious signal value with other values*. This keeps the PID controller in a region of operation for which its behaviour is predictable. This may not ensure correct *system* behaviour, but it may be sufficient protection e.g. against noise spikes on sensor readings, given the robustness of the PID control law. With a slight complication, the wrapper could be specified to set p to its last previous value, rather than 0, to reduce the step change in the input to the OTSC.

In many cases, though, it is not judged useful to correct a "suspicious" input value to the ROS or the OTSC. It may still be possible to prevent harm by checking and if necessary correcting their subsequent outputs. Suppose that a failure causes "suspicious" values of p. The designer may decide that the wrapper will then perform additional plausibility checks on the output of the PID controller. If the checks fail, the wrapper could ensure *graceful degradation* by providing a simpler version of the OTSC's (or ROS's) function. The designer might specify this kind of switch if the degraded control were proven to keep the boiler in an acceptable degraded mode of operation for as long as the OTSC cannot be trusted to perform correctly.

All these palliative measures may only be acceptable for a short time. If they persist, a reaction can be for the wrapper to enforce at least safe system-level behaviour, by switching the burner off ($BC' = 0$): an extreme form of graceful degradation suitable for all undesired situations.

Another possibility is *error recovery*. In many OTSCs, after most failures a reset is sufficient to restore an internal state such that the OTSC will subsequently exhibit correct (nominal) behaviour. In our example, the wrapper could reset the PID controller (OTSCS) if its output is clearly out of bounds. Reset erases the OTSC's memory of previous history: it does not generally guarantee that its future behaviour will be appropriate *from a system viewpoint*, but it may in a control system like our example, if the designer can demonstrate that the internal state of the OTSC will then return to a correct state (through the OTSC reading and processing its inputs) quickly enough.

More complex recovery actions can be specified. If, for instance, an OTSC has an "undo" operation, the wrapper could use it for *backward recovery and retry*; a wrapper could store sequences of input messages to an OTSC and replay them after recovery, possibly even with slight variations to reduce the risk of repeated failure ("retry blocks" architecture [2]). The possibilities here are bounded by the risk implicit in increasing the complexity of the wrapper, and thus the risk of specification or implementation errors. For instance, designers may often limit themselves to stateless wrappers.

The case of reset is an example of a wrapper generating *exception* signals rather than just manipulating the normal ROS-OTSC communications. As an-

other example, the wrapper can generate an exception signal to the ROS, E_{TTH}, when e.g. the OTSC's BC' output, or the T reading, exceeds specified bounds.

Last, many of the actions described so far may not be effective, e.g. if the cue to which they react is caused by a permanent or recurrent fault. If this is considered too likely, wrappers may be designed to escalate to more drastic and safer actions (multi-level recovery). E.g., once it has entered a "graceful degradation" state, a wrapper could become sensitive to cues that it would otherwise ignore, and trigger a more drastic action if any of these cues occurs. After the wrapper has reset the PID controller, it may set a time-out after which it will shut down the boiler if normal control has not resumed. Again, designers need to judge at which point the added complexity becomes counterproductive.

6 Probabilistic Dependability Properties

Up to this point, we have approached wrapper design mostly from a deterministic viewpoint: the designer considers the possibility of certain unplanned-for sequences of actions of the OTSC or ROS, and specifies the wrapper so that it will mask or alter those behaviours in ways that appear desirable, to achieve one of the specified nominal or degraded modes of operation.

This desirability must be determined in view of the system-level dependability requirements, which are inevitably, in their general form, probabilistic, as outlined in Section 3.

A wrapper may be meant to avoid or mask certain component failures, or to mitigate them; it may improve system dependability by avoiding certain system failures, i.e. increasing the probability of nominal behaviour, or by mitigating them, i.e. shifting some probability from more severely to less severely degraded behaviours.

As always with fault tolerance, wrapping faces two kinds of trade-offs, i.e. between the improvement in dependability that it produces by avoiding or mitigating some failures, and (i) its direct costs, in development effort and in terms of run-time resources; and (ii) the dependability loss that wrappers cause by *causing* failures or making them more severe.

Costs are generally the easiest factor to estimate. Estimating dependability improvements may be difficult. In some cases, system failures due to specific failure modes of OTSCs are observed often enough that it is easy to assess the expected effect of avoiding them (and to determine how to). But if a system is already reasonably dependable without wrapping, the dependability gain will be uncertain. Even so, designers will think it reasonable to provide abilities at least to deal with predictable component failures that have a clear potential for severe effects and can be avoided or tolerated at low cost. This appears to be the approach, for instance, of the HEALERS project [5]. However, this common sense approach, when extended to less obvious failures, is not guaranteed to improve dependability, due to difficulties with the second trade-off.

As said in Sec. 4, interventions by wrappers generally substitute a controlled degraded system behaviour (a more acceptable failure) for a potentially uncon-

trolled failure. The designers decide to which cues the wrapper reacts. Including more cues avoids more uncontrolled failures, but also causes more wrapper interventions on "false alarms", *causing* degraded behaviour when nominal behaviour would otherwise occur. Designers cannot *a priori* judge which occurrences of a given cue are false alarms, and thus whether, statistically, wrapper intervention on that cue improves dependability. Besides, in many systems the effects of wrapper interventions on the behaviour of the whole system will be more complex to trace than in our boiler example.

A wrapper may also cause system failures in the the obvious way, because of bugs or physical faults, and deliver, for instance, a wrong input for the ROS despite having received a correct OTSC output; or, for the same reason, not react to a cue as specified. For many systems this risk will be negligible, however, because the wrappers will be simple and easy to verify, compared to the risk of either "false alarms" or failures to intervene that are directly due to the designers' choices. That is, most wrapper failures will be due to the inherent limits of the algorithms that a designer can feasibly apply. Error detection, for example, often depends on reasonableness checks, which cannot flag values that are erroneous but "reasonable". They can be made more stringent at the cost of using cues that are not sure indications of errors. Designers thus know how to shift the balance between false alarms and uncontrolled failures, and can even choose which component failure modes the wrappers will not detect or tolerate, and in which circumstances they may produce false alarms. Unfortunately, they still do not usually know the frequency of these events, so that the uncertainty on the actual dependability improvement achieved by wrapping is not resolved.

Design faults in wrappers remain a potential problem in the case of more complex wrappers. Designers must decide how sophisticated a wrapper they can specify to be before this very sophistication becomes counterproductive. To make the transition less sharp, it may be worthwhile to seek wrapper design techniques that bias wrappers towards "benign" failures, whose consequences can be assessed, rather than uncontrolled ones, like injecting arbitrary values in a communication stream.

7 Conclusion

We have tried to clarify some issues concerning *protective* wrapping. Protective wrappers are components that monitor and ensure the non-functional properties at interfaces between components. We have described the role that protective wrapping may play as a special case of fault-tolerant design, from both the viewpoints of deterministic and of probabilistic dependability properties.

These considerations should help designers in specifying wrappers, using the spectrum of fault-tolerance techniques within the special constraints of wrapping as a design structuring scheme. These peculiarities are not always acknowledged in previous literature. The main considerations we have made are: wrappers can be rigorously specified on the basis of the designers' specification of the OTSC's behaviours in its possibly multiple modes of operation: from nominal,

correct behaviour to manageable, non catastrophic failure modes. Also, due to poor documentation and poor ability to detect run-time errors inside off-the-shelf components, protective wrappers may have to act on "cues" of potentially erroneous and/or error-causing communications between components. All of this increases the importance of design trade-offs between reducing the probabilities of the more dangerous system failure modes and avoiding too frequent false alarms leading to degraded service or "safe" system failures.

Research developments that appear desirable concern formal proof, probabilistic modelling and experimental evaluation. Formal proof methods, tailored to the restricted sets of structures defined by wrapping and the kinds of properties it involves, are desirable to support the verification steps described in Sec. 3. Probabilistic modelling should support designers in choosing trade-offs as discussed here; it must cover both the structural aspects of how component failures cause system failure, aspects that are well developed in modelling of fault tolerance, and the uncertainty on the reliability of the individual components and their probabilities of failing together, as studied in software reliability research and the assessment of software diversity. Last, experimental evaluation of systems using protective wrapping is required, to document the ranges of error coverage levels, "false alarm" rates and system dependability achieved with various classes of wrapper designs and of OTSC components and thus give some basis for informing probabilistically based decisions.

Acknowledgements

This work was supported in part by the U.K. EPSRC through project DOTS (Diversity with Off-The-Shelf Components), GR/N24056.

References

1. T. Anderson, M. Feng, S. Riddle , A. Romanovsky, *Protective Wrapper Development*, Proc. 2nd Int. Conf. on COTS-Based Software Systems, Ottawa, Canada, 2003.
2. P. E. Ammann , J. C. Knight, *Data Diversity: An Approach to Software Fault Tolerance*, IEEE TC, C-37, pp. 418-25, 1988.
3. J. Arlat, J.-C. Fabre, M. Rodriguez , F. Salles, *Dependability of COTS Microkernel-Based Systems*, IEEE TC, C-51, pp. 138-63, 2002.
4. S. Cheung , K. N. Levitt, *A Formal-Specification Based Approach for Protecting the Domain Name System*, Proc. DSN 2000, International Conference on Dependable Systems and Networks, New York, USA, 2000.
5. C. Fetzer , Z. Xiao, *HEALERS: A Toolkit for Enhancing the Robustness and Security of Existing Applications*, Proc. DSN 2003, International Conference on Dependable Systems and Networks, San Francisco, U.S.A., 2003.
6. T. Fraser, L. Badger , M. Feldman, *Hardening COTS Software with Generic Software Wrappers*, Proc. 1999 IEEE Symp. on Security and Privacy, Oakland, CA , USA, 1999.

7. A. K. Ghosh, M. Schmid , F. Hill, *Wrapping Windows NT Software for Robustness*, Proc. 29th IEEE International Symp. on Fault-Tolerant Computing (FTCS-29), Madison, USA, 1999.
8. B. Meyer, *Applying "Design by Contract"*, IEEE Computer, 25, pp. 40-51, 1992.
9. P. Popov, L. Strigini, S. Riddle , A. Romanovsky, *Protective Wrapping of OTS Components*, Proc. 4th ICSE Workshop on Component-Based Software Engineering: Component Certification and System Prediction, Toronto, 2001.