NEW DESIGN TECHNIQUES FOR ENHANCING FAULT TOLERANT COTS SOFTWARE WRAPPERS

Luping Chen and John May
Safety Systems Research Centre, University of Bristol, Bristol, BS8 1TR, UK

Keywords: Safety critical system, COTS software, fault tolerance, wrapper.

Abstract: Component-based systems can be built by assembling components developed independently of the systems. Middleware code that connects the components is usually needed to assemble them into a system. The ordinary role of the middleware is simple glue code, but there is an opportunity to design it as a safety wrapper to control the integration of the components to help assure system dependability. This paper investigates some architectural designs for the safety wrappers using a nuclear protection system example. It integrates new fault-tolerant techniques based on diagnostic assertions and diverse redundancy into the middleware designs. This is an attractive option where complete trust in component reliability is impossible or costly to achieve.

1 INTRODUCTION

Commercial-off-the-shelf (COTS) software based components are increasingly being included within complex safety critical systems (Profeta et al 1996). Therefore it is vital both to distinguish adequate software components from inadequate ones as well as to determine the effect on system dependability of replacing previous systems with COTS software based components. Yet the know-how to construct dependable safety critical applications from dependable COTS software based components remains the 'holy grail' within the area of component-based software engineering (Crnkovic and Larsson 2002).

The component-based software development process is based on the reuse and integration of high-level software components and bespoke components to form a new system (Brown and Wallnau 1998). A key issue for such hybrid systems is to show that the use of the software components (which will be considered as 'black boxes') does not compromise the safety, reliability and (perhaps) security of the overall system, since the reliability of software components cannot be fully assured prior to integration (Voas 1998). Furthermore, even if such pre-assurance was a theoretical possibility, it would seldom be available, since the software components are commonly developed to unknown standards or standards aimed at general use, which are insufficient for safety applications (Profeta et al 1996, Lindsay et al 2000). These difficulties are sometimes compounded by the inaccessibility of some COTS code. Where examination of code is not permitted, traditional assurance techniques (with the exception of black box testing) cannot be applied to COTS components post-purchase, to supplement the supplier's verification and validation (V&V) activities. In general it is necessary to use 'middleware', possibly based on standard infrastructure technologies to integrate the inevitably disparate components (May 2002). This middleware can also be used to play the role of a safety wrapper, and offers an important opportunity to include component adaptation and monitoring strategies, to help ensure the overall system's dependability (Shin and Paniagua 2006). One significant use of these wrappers is to enable the replacement one COTS software component of a safety critical system by another without significantly modifying the wrapper itself: for example, an upgraded component. The wrapper in this case ensures correct behavior over key safety aspects of the components functionality.

This paper develops some design techniques for enhancing fault tolerant COTS software wrappers. The underlying concept of software fault tolerance assumes that any system has unavoidable and undetectable software faults no matter how thoroughly the software has been debugged, modularised, verified and tested. Hence programming strategies to prevent or recover from software failures must be included within a complex safety critical system such that it can provide service even in the presence of software faults. Current programming strategies are classified as N-Version Programming, Recovery Block and Self-Checking Version Schemes. Some new methods have been developed for improving software fault tolerance based on diverse redundancy and diagnostic assertions (Napier 2001, Chen et al 2002), and these designs can be assessed by fault injection techniques. An approach to assessment using Perturbation of Interface Parameters (PIP) of COTS components has been developed to simulate a range of internal COTS component faults (Chen et al 2004).

Based on the example of using smart sensors as COTS components within a plant protection system, this paper considers new diversity and diagnosis strategies for safety wrapper design in COTS-based systems, together with methods for assessing their effectiveness.

2 FAULT-TOLERANCE DESIGNS

2.1 Fault Tolerance by Assertion

The use of assertion/diagnostics is often based on a rather restricted view of failures. Traditional approaches to fault detection in software often focus on specific *anticipated* problems, usually those that halt the program execution. However this view is far from general because anticipated problems are a small subset of all possible problems. The more subtle and interesting failures are quite different and not addressed by these traditional approaches. Such failures are caused by errors in the design of the underlying program algorithms - i.e. it is the proposed solution to the problem that is flawed, not the implementation of the solution (Harel 1992). Failures caused by these *algorithmic* errors do not necessarily halt the program execution, they simply compute an *incorrect answer* (Napier 2001). Anticipated problems are easier to detect and contingencies can be put in place to put it into a safe state or initiate an appropriate recovery procedure. Non-halting failures of a COTS component are much more likely to remain unrevealed and if such insidious erroneous states are allowed to propagate from a component into the rest of the system potentially disastrous consequences can result. A wider view of fault detection is required which aims to detect these algorithmic errors in addition to the traditional anticipated problems.

There is a range of on-line diagnostic techniques available including data encoding (Napier 2001). Most approaches to safety wrapper design use a conventional user-defined executable assertions (Napier et al 2000). User-defined assertions can be either external to the original program, based on input/output relationships, or applied to check internal program states. A safety wrapper can be applied to a particular component using assertions in one of three ways: Pre-condition assertion, Post-condition assertion or Point/intermediate assertion.

In general, user-defined executable assertions for inspecting internal data states may be an integral part or an external wrapper for the underlying program. They can be implemented relatively simply by adding extra lines of code, possibly utilising special mechanisms provided by the high level language. Assertions requiring access to program variables that are not accessible at the components I/O, are unsuitable for use with COTS software since these variables are not visible outside the component. In contrast, external assertions can be integrated into the middleware of component-based systems even if a program component needs to be treated as a black box.

2.2 Fault Tolerance by Diversity

Early ideas for reliability improvement by diversity design centred on multiple versions of software fulfilling the same requirement specifications. The versions are expected to show different (diverse) failure behaviours, both in terms of the inputs that cause them to fail and in terms of failure behaviour when both versions fail at the same time, so that discrepancies between the two versions flag failures. Currently, the main strategy for building versions with such diverse behaviour is to use developers with different backgrounds or to force diversity by use of different hardware, languages, compilers etc. The hope is that the different versions will not contain the same errors. However, practical applications of diverse software have shown that normal design methods can not be assumed to achieve this goal. Certainly, assuming independent failures in versions will often be optimistic (overestimate reliability).

Two kinds of diversity can be considered in middleware designs:

Structural diversity: An algorithm can be implemented with different structures.

Functional diversity: a specification can be fulfilled by different algorithms.

Structural diversity has to be abandoned as a technique for achieving diversity between a wrapper and a COTS component, since the internal software design in a COTS component may be unknown (Chen and May 2004). However, structural diversity can be used in a multi-version approach to wrapper design. In a multi-wrapper design, the reliability of a system with a COTS component will depend on the combined failure coverage of all safety wrappers. The diversity between wrappers will be an important factor. There is still a need for diversity between wrappers and the COTS component; to be effective, the wrapper should not fail (to trap the COTS failure) at the same time as the COTS component fails. This is a black-box issue and can not be judged/assessed directly. But it seems plausible to expect that if wrapper A is not diverse with the COTS component and wrappers A and B are diverse, then wrapper B *will* show some diversity with the COTS component. Thus a potential strength of a multi-wrapper approach is that the diversity/reliability improvement issues are judged on the basis of wrapper design and there is less emphasis on the COTS design, about which little may be known.

2.3 Data Diversity

Data diversity generally can be regarded as a special assertion/diagnostic technique defending against design faults. The rationale of this technique is that failures of defective software are usually input dependent, e.g. the faults contained in the software can only be triggered by fixed input sequences (Ammann & Knight 1988). The key technique is to use re-expression of an original input in 'retries' that use the same software code. In one form of the technique, the goal of the retry executions is to use these different inputs to generate outputs that can be manipulated to re-create the correct output for the original input. This provides two (or more) computations of the required output based on different inputs, so there is a reduced probability of failure.

Data diversity is a diverse software technique where different multi-versions use the same code, but use re-expressions of the input, but it might be possible to pre-empt this approach and incorporate some degree of execution flexibility into the design of software components to simplify the use of the data diversity concept. In this paper, the data diversity technique could be used in conjunction with safety wrappers or in fault tolerance functions within the component).

3 CASE STUDY PROGRAM

The case study used the protection system for a power plant from a multi-version software diversity project known as DARTS (Quirk & Wall 1991). A single C version of the software was used in the experiments. The DARTS software was developed specifically for experimentation, but this was done under commercial conditions to produce software representative of that used in practice. The program monitors various parameters from the plant environment: neutron pressure (NP), steam drum pressure (SDP) and steam drum level (SDL), and sets one of seven output trIp signals together with several status signals under various circumstances. Each of the physical parameters is monitored by triplicated Sensor Devices**.**

The signals received from the smart sensor can not be used directly by the protection system. A middleware module ASSIGN_VALUE has been designed to transfer the sensor values into proper formats accepted by the protection system. This software module also plays an important role as a safety wrapper, attempting to check the correctness of the input values from the smart sensors using consistency checks.

In its role as 'safety wrapper,' ASSIGN_VALUE is a common module monitoring all three physical parameters NP, SDL and SDP. It receives three readings for a physical parameter, and checks their compatibility by observing if the differences between the values from the three sensors are within allowed ranges (specified in the system requirements). Then it will decide how to pass on the raw input data from the sensors and set the data statuses.

For example, one assertion/diagnosis function in ASSIGN_VALUE is to check the three input values then decide whether:

- All three values are averaged
- Only low and medium values are averaged
- Only high and medium values are averaged
- A valid value can not be assigned

The protection system then computes using the pre-processed values output from wrapper ASSIGN_VALUE. ASSIGN_VALUE was used in a range of experiments measuring diversity and fault coverage, as follows.

3.1 Wrappers with Structural Diversity

Two distinct factors influencing software diversity, structural coupling and fault distribution, were identified in recent research [Chen et al 2002]. It was also shown that there is scope to manipulate these factors in practice to actively improve the diversity of software versions. Therefore, for example, if a safety system design were to use diverse software-based sensors, the implementation should choose COTS sensors that are, or can be configured to be, naturally diverse according to these factors. This has the potential to strengthen safety cases for systems containing diverse channels.

One important structural factor for diversity enhancement is to implement 'orthogonal structures' in versions e.g. a two-dimensional input-space can be processed: {For X=1 to N then {For Y=1 to N then OP1}} or orthogonally {For Y=1 to N then {For X=1 to N OP2}. This idea has been used to redesign a new wrapper ASSIGN_VALUE_New. Then two version-pairs were constructed for the purpose of assessing diverse designs: A pair with two versions of the original wrapper (original-original) and a pair with two versions of the new wrapper (New-New). The diversity between a version pair is a measure of the difference between their failure behaviours when the two versions are injected with different faults. Using identical versions with different faults simulates the case where two versions are developed by different developers that use the same structure (or at least largely similar structures since bugs can alter structure), but containing different bugs.

Four systems were compared using: the original single wrapper, the new single wrapper, the original-original wrapper pair and the new-new wrapper pair.

An empirical approach to diversity assessment was employed in which diversity is measured based on a sample of injected faults (Chen et al 2002). The resulting diversity assessment measures the average behaviour of the software over a range of fault pairs in two versions. This process was used to assess the diversity present in multi-safety wrappers. That is, to measure their degree of independence (avoidance of common failures) when each safety wrapper contains one fault, and averaging this measured diversity over the range of fault pairs used.

To assess the effectiveness of the fault tolerant designs, the proportion of the faults caught by the safety wrappers will be used as an index. A safety wrapper is designed with the requirement to protect system from failures in a software component, and the quality of safety wrappers can be directly measured by *fault coverage*. The approach is to simulate a number of failures of the software component and then to check how many faults can be detected by the safety wrapper (Panel 2002). This experiment used a test support tool to simulate 512 failure scenarios of the COTS component. For each system, the proportion of the 512 failures that was not trapped by its wrapper(s) was observed (called a *failure probability* in the following results).

In summary, the tasks involved are:

1. Observing the system behaviours with different faults in the multiple wrappers to assess the diversity of the wrappers
2. Observing the system behaviour with faults in the software component and wrappers to estimate the improvement of fault-tolerant ability due to multiple safety wrappers.

The main burden of these assessments is the selection and simulation of representative faults in the COTS component and in the safety wrapper by means of PIP (perturbation of interface parameters) and SFI (software fault injection) methods respectively. Use of SFI to simulate faults inside a safety wrapper is a normal fault injection approach. Applying PIP to simulate faults inside the component has been specially developed to simulate fault injection where the COTS software must be treated as a black box (Chen 2002) i.e. where fault injection is not possible. The safety designs of the wrapper/s aims to catch all potential failures propagated from a COTS component. So the PIP approach needs to simulate all possible types of anomalies of the interface parameters that could be generated by the COTS code.

One result was that the new wrapper pair showed a lower failure probability than the original pair. The new wrapper design does appear to provide a better structure for avoiding common failures. The reduced coupling of the functions has made the faults distribute uniformly over the structure and over the input space, affording fewer opportunities for different faults to be triggered by the same inputs. Moreover, the use of multiple wrappers can improve the level of fault tolerance achieved. The results are summarized in figure 1.

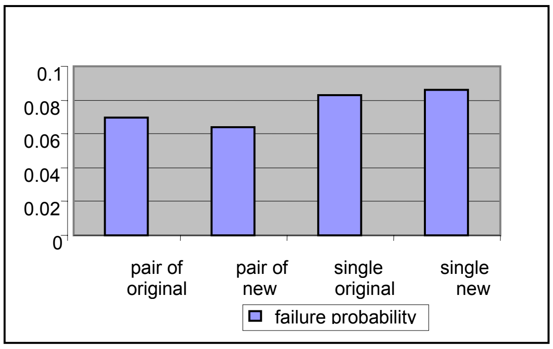

Figure 1: Failure probabilities of different systems.

The systems with multiple wrappers showed higher fault tolerance than those with a single one.

The system containing the wrapper pair with a higher diversity showed the best fault tolerance.

3.2 Wrappers with Functional Diversity

The aim of designing an effective wrapper in this experiment was to enhance its reliability using functional diversity.

In a practical design, a safety wrapper may contain some productive computation functions i.e. that perform computation of values rather than just checking relations between values. Therefore in addition to a structural diversity approach, we can use functional diversity in the design of multiversion wrappers. Essentially, the idea is to build a 'checking style' wrapper for the initial wrapper — thus 'wrapping a wrapper.' For example, the safety wrapper *ASSIGN_VALUE* includes some productive functions to calculate the average value of three sensor readings under various conditions. Our approach is to use different diagnosis strategies in wrappers to check if the calculation of the average value is accurate.

In the experiment, a 'checking style' wrapper was built as a redundant assertion. With the original wrapper, they can compose a two-version wrapper array. This check style wrapper used reverse computation to validate the output from the COTS component or the previous wrapper. The assessment simply compared the fault coverage of the diverse wrappers with the original one. We use a subset of faults (from the fault set in last experiment) to test the two wrappers. Of these faults, the system shows diversities on 22.2% of them.

3.3 Wrappers with Data Diversity

Application of data diversity to smart sensors relies on finding an equivalent re-expression of the input-output relation of the sensors.

A smart sensor is normally used as a COTS component. This means that it is difficult to implement diversity design because little is known about the code inside the component. On the other hand, smart sensors are often well suited to data diversity due to their straightforward dataflow.

The functions of a smart sensor mainly include Data Acquisition and Signal Processing. The processor may be also used for data conversion. The specification of a smart sensor is often to measure a variable, manipulate it and, in some cases, to take action based on its value. In many wider applications involving smart sensors, one does not care about the raw data, but only about the information derived from it by the sensor. For example in the DARTS protection system, the application may not need the exact temperature or pressure, but is interested in whether it has exceeded a certain threshold or not. Instead of sending a continuous stream of temperature or pressure readings, a smart sensor would send just one message when the temperature criteria are met. Thus, only the relevant information is sent out to the surrounding system.

Similarly to design strategies using functional diversity, data diversity techniques can be integrated in the safety wrapper based on origin-point shifting methods. To process an input variable, smart sensors mainly use functions such as scaling, trimming, and filtering. Input values within different ranges and with different amplitudes of noise have been specified to be handled by different formulas. The above discussion suggests a realistic approach to use of data diversity within a smart sensor i.e. treating it as a white box. If a smart sensor is a black box, it is difficult to set up a proper post-condition assertion without knowledge about its filter, calibration or linearization functions. But we often know the function of the smart sensor essentially is monotonic:

Suppose f is the function of the smart sensor such that: $output = f(input)$ and D is its defined input domain. Then we have the relation: $\forall$ inputs $x, y \in D$, if $x > y$ then we have $f(x) > f(y)$.

To improve the safety of a smart sensor being used in an alarm function, we can design data diversity by re-expressing the trip points or other thresholds using device configuration facilities. For example, we can shift both the input value and trIp point by a value *L* at the same time. The shifted values are used as re-expressed inputs and fed into the sensor. In our solution, the sensor component was wrapped with a new checker to judge if the trip criteria were met based on these shifted values. Thereafter, an assertion simply compared this result with original one.

4 CONCLUSIONS

This paper investigates some design strategies to provide middleware with safety functions to enhance dependability of systems containing COTS components. The designs were evaluated using an empirical method and support tool developed systematically for diversity estimation based on fault injection and failure tracing. A key issue is the assessment of fault tolerance, on which conclusions are based. The presented approach used fault injection. By definition these faults are artificial. The current understanding of software failure modes is insufficient to allow the definition of *realistic* fault sets - it is not clear what these might be. Therefore this approach relies on an assumption that hypothetical fault sets used in this way are informative.

Diverse wrappers and wrappers embedded with diagnostic assertions or data diversity have been demonstrated as providing some level of increased effectiveness at protecting system from potential defects inside a COTS component.

The experiments on diversity designs of safety features in middleware did not distinguish functional and structural diversity: in terms of performance they were similar. Anyhow, from these limited experimental results it would clearly not be possible to make general claims about the fault detection capabilities of the different assertion type.

Some tentative conclusions are suggested that are relevant to practice:

- A ‘wrapper’ can be built from multiple smaller complementary wrappers which can be very effective and easy to implement
- Functional diversity is easier to design than structural diversity in multi wrappers
- The application of check-style wrappers reduces the scope for faults because they are usually simpler modules than other kinds of functional wrappers. It was clear in our experiments that check-style wrappers can be considerably more succinct than the code they check. This is not surprising; it is well known that checking a function can be a less complex task than computing it. This effect was sometimes so pronounced that it was difficult to select plausible fault modes for injection into the check-style wrappers.
- A degree of orthogonality between old and new wrappers was observed, which suggests that software reliability will be most improved if both assertion types are used (particularly for faults with small footprints in the input space).
- Data diversity would appear to offer an effective and appropriate way to improve safety in smart sensors. It remains unexplored in practice.

ACKNOWLEDGEMENTS

The work presented in this paper comprises aspects of a study (NewDISPO2-4) performed as part of the UK Nuclear Safety Research programme, funded and controlled by the CINIF together with elements from the SSRC Generic Research Programme funded by British Energy, Lloyd's Register, and the Health and Safety Executive.

REFERENCES

Ammann, P.E., Knight, J.C., 1988. Data Diversity: An Approach to Software Fault Tolerance. *IEEE Trans. on Computers*, 37(4): pp. 418-425.

Brown, A., Wallnau, K., 1998. The Current State of CBSE. *IEEE Software*, 15(5): pp.37-46.

Chen, L., May, J., Hughes, G., 2002. Assessment of the Benefit of Redundant Systems, *Lecture Notes in Computer Science, volume 2434, Springer, pp.151-162.*

Chen, L., May, J., 2004. Safety Assessment of Systems Embedded with COTS Components by PIP technique, *Lecture Notes in Informatics 58 GI.*

Crnkovic, I., Larsson, M., 2002. *Building Reliable Component-Based Software System,* Artech House Books.

Harel, D., 1992. *Algorithmics: The Spirit of Computing,* Addison-Wesley.

Lindsay, P., Smith, G., 2000. Safety Assurance of Commercial-Off-The-Shelf Software, *Proc 5th Australian Workshop on Safety Critical Systems and Software*.

May, J., 2002. Testing the reliability of component-based safety critical software. *Proc. 20th International System Safety Conference, pp. 214—224.*

Napier, J., Chen, L., May, J., Hughes, G., 2000. Fault Simulating to validate fault-tolerance in Ada. *International Journal of Computer Systems, 15(1):61-67*

Napier, J., 2001. *Assessing Diagnostics for Fault Tolerant Software.* PhD thesis, Department of Computer Science, University of Bristol.

Panel Discussion, 2002. How useful is software fault injection for evaluating the security of COTS products. *Proceedings of the 17th ACSAC, IEEE Computer Society*.

Profeta., J, Andrianos, N., Yu, B., 1996. Safety-Critical Systems Built with COTS, *IEEE Comp. 29(11), pp 46-54.*

Quirk, J., Wall, N., 1991. Customer Functional Requirements for the Protection System to be used as the DARTS Example, *DARTS consortium deliverable report DARTS-032-HAR-160190-G supplied under the HSE programme on Software Reliability.*

Shin, M., Paniagua, F., 2006. Self-Management of COTS Component-Based Systems Using Wrappers, *30th COMPSAC, pp. 33-36.*

Voas, J., 1998. Certifying Off-The Shelf Software Components, *IEE Computer, pp.53-59*.