RUPRECHT-KARLS-
UNIVERSITÄT
HEIDELBERG

International Workshop on the
Design of Dependable Critical Systems

ECOMODIS

Proceedings

of the
International Workshop on the
Design of Dependable Critical Systems
"Hardware, Software, and Human Factors in Dependable System Design"

DDCS 2009

September 15, 2009
Hamburg, Germany

In the framework of
The 28th International Conference on
Computer Safety, Reliability and Security
SAFECOMP 2009

Edited by

Achim Wagner[1], Meike Jipp[1], Colin Atkinson[2] and Essameddin Badreddin[1]

[1] Automation Laboratory, Institute of Computer Engineering, University of Heidelberg
[2] Chair of Software Engineering, University of Mannheim

International Workshop on
the Design of Dependable Critical Systems
September 15, 2009, Hamburg, Germany

Quantifying Safety in Software Architectural Designs

Atef Mohamed and Mohammad Zulkernine

School of Computing
Queen's University, Kingston
Ontario, Canada K7L 3N6
{atef, mzulker}@cs.queensu.ca

Abstract. Incorporating safety in the software architectural design decisions is important for the successful applications in safety-critical systems. However, most of the existing software design rationales do not consider the quantitative aspect of the software architectures with respect to safety. As a result, alternative architectures cannot be compared adequately with respect to safety. In this paper, we present an analytical approach for quantifying safety in software architectural designs. We use the concept of architectural service routes to quantify system safety in terms of software architectural attributes. We show how to make appropriate architectural design decisions based on their impacts on safety. We compare different example architectures with respect to system safety.

Key words: Software architecture, architectural design decisions, and system safety.

1 Introduction

Appropriate architectural design decisions are important for achieving quality attributes in software intensive systems. These decisions are to be taken in the early design stages and their impacts are carried out among the later development stages. *System safety* is the absence of catastrophic consequences on the system user(s) and the environment [1]. In safety-critical systems, failure types differ with respect to their criticalities (catastrophic impacts) [5]. For example, a traffic light system is highly critical to content failure (incorrect service), where the traffic lights are green in all directions. On the other hand, it is less critical to silent failures (service stopping), where all lights are turned off. An aircraft control system is more critical to silent failures than a production line control system's criticality to the same failures. Unfortunately, safety has not been sufficiently addressed at the software design level, and the quantitative impacts of software architectures on safety have not been explicitly considered in the existing software architectural design methodologies. As a result, existing architectural strategies fail to sufficiently incorporate the rationale behind the adoption of alternative architectural mechanisms with respect to their impacts on system safety [13].

Few techniques consider software architectural design decisions with respect to their impacts on safety. These techniques mainly provide a set of requirements to achieve system safety [13, 10, 3] or provide safety analysis mechanisms [5, 12]. Weihang Wu *et al.* [13] introduce some software architectural design tactics to consider safety in software architectures. The approach extends existing software architecture design tactics to consider system safety through the appropriate elicitation, organization, and documentation. Swarup *et al.* [10] propose a framework for achieving system safety through system hazard analysis, completeness of requirements, identification of software-related safety critical requirements, safety-constraints based design, runtime issues management, and safety-critical testing. Hill *et al.* [3] identify a number of safety requirements that must be possessed by a system or system component. These requirements are identifiability, stability, completeness, clarity, validity, and feasibility. Leveson *et al.* [5] and Tribble *et al.* [12] provide safety analysis based on architectural designs using Fault Tree Analysis (FTA) mechanism. FTA allows the detection of unsafe computational states and consequently, it prevents safety critical failures. However, current techniques disregard the quantitative evaluation of safety in software architectures that can incorporate system safety through the appropriate selection of the architectural design decisions.

In this paper, we present an analytical approach for quantifying safety of software architectural designs. We evaluate system safety in terms of software architectural attributes using the concept of Architectural service routes (ASRs) [8]. The concept of Architectural service routes allows quantifying architectural quality attributes by viewing a software architecture as a set of components and a set of service routes connecting them. We provide an architectural design decision approach for selecting the appropriate architecture based on its impact on safety. Finally, we compare three different example architectures based on their impacts on safety. We use "Make To Order" manufacturing planning process in our example architectures.

2 Preliminaries

Software architecture of a system is the structure, which comprises software components, the externally visible properties of those components, and the relationships among them [2]. *A component* is a unit of composition with contractually specified interfaces, explicit context dependencies only, and no persistent state. A *component interface* is a mean by which a component connects to another component [11]. A component has one or more *provided* and/or *required* interfaces [9]. *A component service* is a facility that a component provides to, or requires from other components as specified in the formal contracts with these components. Software failures are classified from *failure domain* viewpoint as content, silent, early service delivery, performance, halt, and erratic failures. [1, 6]. We denote the set of all failure types by T. *Failure criticality* is the estimated degree of catastrophic impact by the failure occurrence.

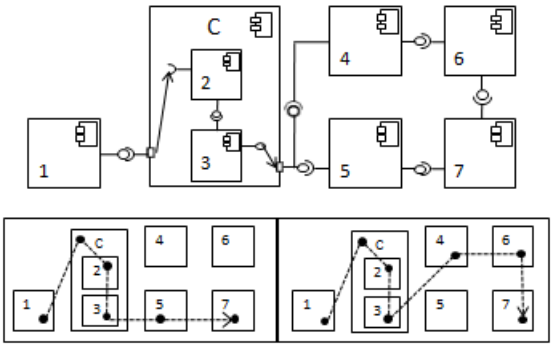


Fig. 1. Architectural service routes of an example architecture

An ASR is a sequence of components that are connected using "provided" or "required" interfaces [8]. Fig. 1 shows some ASRs of an example architecture in UML 2.0. Component 2 provides service to component 3. Component 3, on the other hand, provides services to both components 4 and 5. Therefore, component 2 provides its service to components 4 and 5 indirectly through component 3. In the bottom part of Fig. 1, we show two example ASRs between components 1 and 7 of the provided component diagram. The sequences of components are $(1, 2, 3, 5, 7)$ and $(1, 2, 3, 4, 6, 7)$ for the left and the right ASR, respectively.

Any two components x and y can have 0 or more ASRs. In Fig. 1, components 4 and 5 have 0 ASR, components 2 and 6 have 1 ASR: (2,3,4,6), and components 3 and 7 have 2 ASRs: (3,4,6,7) and (3,5,7). We refer to the set of ASRs from x to y as Ψ^{xy}, and we denote an ASR in this set as ψ_k^{xy}, where k is the index of the *k-th* ASR in Ψ^{xy}. The length of an ASR ψ_k^{xy} (referred as L_k^{xy}) is the number of components in it. In Fig. 1, $L_1^{2,6} = 4$, $L_1^{3,5} = 2$, and $L_1^{4,5} = 0$. $|\Psi^{xy}|$ denotes the number of ASRs from component x to component y *e.g.*, $|\Psi^{3,6}| = 1$ and $|\Psi^{3,7}| = 2$.

3 Evaluating system safety

To derive system safety in terms of architectural attributes, we exploit the results of the failure propagation analysis using ASRs [8]. Failure propagation indicates the probability that a failure propagates through system components. The quantitative evaluation, parameters, and assumptions are described in the rest of this section.

From the combinatorial viewpoint, *system safety* is the non-occurrence probability of failures that can lead to a mishap or hazard, whether or not the intended function is performed [4]. Therefore, system safety S is expressed as $\prod_{f \in T}(1 - \lambda^f p^f)$, where p^f is the probability of occurrence of system failure f, and λ^f is the criticality of failure f [7]. Failure criticality can be estimated based on expert opinion or design documents.

By considering failure propagation in software architectures, a system failure occurs when a component failure is propagated along an ASR to one of the output

interface components. Given that, we can rewrite the safety equation as follows, $S = \prod_{i=1}^{I} \prod_{f \in T} (1 - \lambda^f p_i^f)$, where I is the number of system output interface components, and p_i^f is the probability of occurrence of failure $f \in T$ at the system output interface component i. We can also replace p_i^f by $\sum_{j=1}^{J} p_j^f P_{ji}^f$, where J is the number of system components, p_j^f is the failure probability of component j, and P_{ji}^f is the probability of failure propagation from component j to interface component i. *I.e.*,

$$S = \prod_{i=1}^{I} \prod_{f \in T} (1 - \lambda^f \sum_{j=1}^{J} p_j^f P_{ji}^f) \tag{1}$$

Eq. 1 evaluates system safety based on failure propagation and failure criticality. Failure propagation from any component j to interface component i is calculated in [8] as follows.

$$P_{ji}^f = \sum_{k=1}^{|\Psi^{ji}|} \beta^{2L_k^{ji}|T|} \tag{2}$$

where β is a any value from 0 to 1, which expresses component failure probabilities of system components. $|T|$ is the number of failure types considered in the evaluation. (*e.g.*, $|T| = 3$ to consider content, silent, and performance failures). By substituting from Eq. 2 into Eq. 1, we get the system safety as follows.

$$S = \prod_{i=1}^{I} \prod_{f \in T} \left(1 - \lambda^f \sum_{j=1}^{J} \left(p_j^f \sum_{k=1}^{|\Psi^{ji}|} \beta^{2L_k^{ji}|T|}\right)\right) \tag{3}$$

Eq. 3 shows system safety in terms of the software architectural attributes and failure criticalities.

4 Architectural design decision for incorporating safety

Software designers of safety critical systems often need to select an architecture from a set of alternative architectures based on their impacts on safety. The propagation of safety-critical failures among these architectures directly impacts system safety based on the ASR attributes as shown in the previous sections. In this section, we show how to consider the quantitative evaluation of system safety in the architectural design decisions.

We provide an algorithm for evaluating system safety and selecting the appropriate architecture based on the ASR attributes among system components. Algorithm 1 provides one of the following decisions to choose between the two architectures A and A'. *SELECT-A* indicates that architecture A is selected, while *SELECT-A'* represents the selection of architecture of A'. *SELECT-EITHER* means that both architectures have equal impact on system safety.

The algorithm allows considering specific failure types in the architectural design decision (Line 1). For example, by considering only content failures, the

Algorithm 1 Architectural design decision based on safety

Input: Architectural attribute values.
Output: Selected architecture.

01. Identify the set of failure types T for comparing A and A'
02. Identify the failure criticality λ^f for each $f \in T$
03. FOR each component j of architecture A DO
04. FOR each output interface component i of A DO
05. Identify the set of ASRs between j and i;
06. END FOR
07. END FOR
08. FOR each component j' of architecture A' DO
09. FOR each output interface component i' of A' DO
10. Identify the set of ASRs between j' and i';
11. END FOR
12. END FOR
13. Calculate safety for architecture A and A' using Eq. 3;
14. IF (safety of $A >$ safety of A') THEN RETURN *SELECT-A*;
15. IF (safety of $A <$ safety of A') THEN RETURN *SELECT-A'*;
16. IF (safety of $A =$ safety of A') THEN RETURN *SELECT-EITHER*;

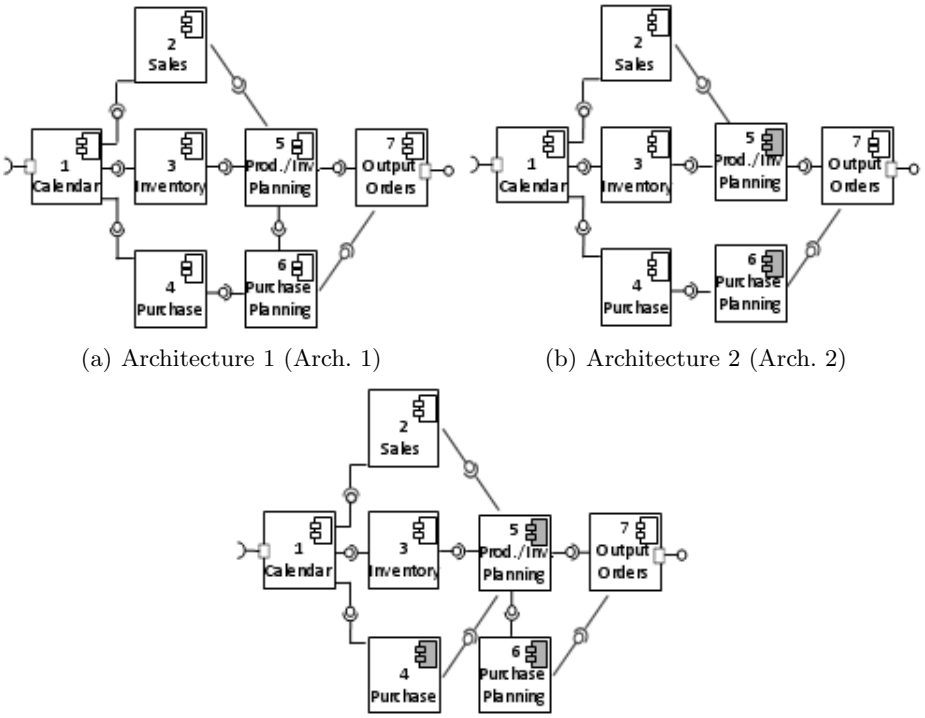
approach will compare software architectures based on data corruption among their component interactions. By considering early service delivery and late service delivery failures, the architectures will be compared based on their performances. Algorithm 1 selects the architecture that has the higher safety value quantitatively. In Line 2, the failure criticalities are identified for the failure types in the set T. These failure criticalities can be identified based on expert opinion or design documents. Lines 03-07 calculate the failure propagation probabilities between each pair of components for architectures A. Similarly, Lines 08-12 calculate the failure propagation probabilities for architectures A'. Line 13 calculates the safety of architecture A and A'. Based on the quantified safety of A and A', Lines 14-16 select the architecture with the higher safety.

5 Case study: comparing safety of example architectures

We use the example of the "Make To Order" (MTO) production planning process of manufacturing systems to explain the proposed technique for comparing different architectures. In MTO, products are manufactured after a confirmed sales order is received for them. We present three different example architectures for this process in Fig. 2. We evaluate the safety of these architectures based on their ASR attributes. Each of the architectures in Fig. 2 uses 7 components, numbered from 1 to 7. Component 1 is an input interface component, in which the user inputs the production planning intervals. It passes the planning intervals to three other components (sales, inventory, and purchase orders) after checking the manufacturing schedule according to the calendar. Component 2 and component 3 deliver the corresponding sales orders and item inventory to the production and inventory planning component 5. Component 4 delivers the purchase orders to the purchase planning component 6. Component 5 also delivers the planned inventory requirements to component 6. Finally, both component 5

and component 6 deliver their outputs to component 7 to create inventory outbound, purchase orders, and planned production orders.

The three architectures differ slightly in the interface with respect to the shaded components. Unlike Fig. 2(a), Fig. 2(b) does not have a connector between components 5 and 6. Fig. 2(c) differs from Fig. 2(a) in that the connector between components 4 and 6 is removed, and another connector between components 4 and 5 is added. Regardless of the functional advantages or disadvantages of these changes, we study these three architectures to see their impacts on the overall system safety. We consider three failures (content, silent, and performance failures), *i.e.*, $|T| = 3$. In the computation of safety, we assume $\beta = 0.7$, since smaller values may result in more approximations and less preciseness. For simplicity, we choose $p_j^f = 0.001$ for all components and $\lambda^f = 0.5$ for all types of failures. We use Eq. 3 to obtain the system safety.



(a) Architecture 1 (Arch. 1)

(b) Architecture 2 (Arch. 2)

(c) Architecture 3 (Arch. 3)

Fig. 2. Different example architectures of "Make To Order" production planning.

Here, we show how to obtain the ASR attributes using Arch. 1 as an example. We also show how to use these attributes to calculate system safety. Since component 1 is an input interface component and component 7 is an output interface component, there is no interface connection from any component to component 1 or from component 7 to any other component.

Table 1.*a*, 1.*b*, and 1.*c* correspond to Arch 1, 2, and 3 respectively. In each

table, rows represent component numbers from 1 to 6 and columns represent component numbers from 2 to 7. The table cells represent the ASR attributes among components in the form (N_1xM_1, N_2xM_2,...), where N_ixM_i means: there exist N_i ASRs of length M_i. For example, a cell (row= 1, column= 5) of Arch. 1 has the value 2x3 since $\Psi^{1,5} = 2$ and both $L_1^{1,5}, L_2^{1,5} = 3$. Similarly, cell (row= 1, column= 7) of Arch. 1 represents $\Psi^{1,7}$ and has the value of 3x4, 2x5. $\Psi^{1,7}$ includes 5 ASRs as follows, $\Psi^{1,7} = \{(1,2,5,7), (1,3,5,7), (1,4,6,7), (1,2,5,6,7),$ and $(1,3,5,6,7)\}$ for $\{\psi_1^{1,7}, \psi_2^{1,7}, \psi_3^{1,7}, \psi_4^{1,7},$ and $\psi_5^{1,7}\}$, respectively. The lengths of the ASRs are $4, 4, 4, 5,$ and 5, respectively. By considering the ASR attributes in Table 1 and the previously mentioned values of p_j^f, β, λ^f, and $|T|$ in Eq. 3, we get, $S = 0.998887872$ for Arch. 1 where $S \in [0, 1]$.

	2	3	4	5	6	7
1	1x2	1x2	1x2	2x3	1x3,2x4	3x4,2x5
2	0	0	0	1x2	1x3	1x3,1x4
3	0	0	0	1x2	1x3	1x3,1x4
4	0	0	0	0	1x2	1x3
5	0	0	0	0	1x2	1x2,1x3
6	0	0	0	0	0	1x2

(a) Arch. 1

	2	3	4	5	6	7
1	1x2	1x2	1x2	2x3	1x3	3x4
2	0	0	0	1x2	0	1x3
3	0	0	0	1x2	0	1x3
4	0	0	0	0	1x2	1x3
5	0	0	0	0	0	1x2
6	0	0	0	0	0	1x2

(b) Arch. 2

	2	3	4	5	6	7
1	1x2	1x2	1x2	3x3	3x4	3x4,3x5
2	0	0	0	1x2	1x3	1x3,1x4
3	0	0	0	1x2	1x3	1x3,1x4
4	0	0	0	1x2	1x3	1x3,1x4
5	0	0	0	0	1x2	1x2,1x3
6	0	0	0	0	0	1x2

(c) Arch. 3

Table 1: ASR attributes of architecture 1, 2, and 3.

Similarly, based on the ASR attributes of Arch. 2 provided in Table 1, the system safety for Arch. 2 is 0.998908816. Comparing the safety values of Arch. 2 and Arch. 1, we can conclude that the Arch. 2 is safer than Arch. 1. This safety gain in Arch. 2 is due to the decrease in the number of ASRs from the system components in general to the output interface component. For example, $|\Psi^{1,7}| = 3$ in Arch. 2, while $|\Psi^{1,7}| = 5$ in Arch. 1. The decrease in the number of ASRs between two components decreases the propagation probabilities and consequently increases the system safety. In Arch. 3, we have increased the number of ASRs (*e.g.*, $|\Psi^{1,7}| = 6$ instead of 5 for Arch. 1) and the lengths of the shortest ASRs (*e.g.*, $L_S^{4,6} = 2$ instead of 1 for Arch. 1). According to our analysis, these changes should decrease the system safety.

Based on the ASR attributes of Arch. 3 shown in Table 1, the safety is calculated as $S = 0.998887773$. Comparing Arch. 3 and Arch. 1, the safety is lower for Arch. 3. The lower safety in Arch. 3 is due to the increase in the number of ASRs among system components. Comparing Arch. 3 and Arch. 2, the safety is lower for Arch. 3. This loss of safety is also due to the increase in the number of ASRs among system components.

6 Summary and future work

Safety has not been sufficiently addressed and the quantitative impacts of software architectures on this quality attribute have not been explicitly considered in the existing software architectural design methodologies. As a result, existing architectural strategies fail to sufficiently identify the rationale behind the

adoption of alternative architectural mechanisms with respect to safety. In this paper, we present an analytical approach for quantifying safety in software architectural designs. We evaluate system safety in terms of software architectural attributes using the concept of ASRs. Finally, we provide an architectural design decision approach for selecting the appropriate architecture based on their impacts on safety-critical failure propagation among system components. The main contribution of this work is to provide a quantitative evaluation of system safety based on software architecture in an early design stage of software system development. In our future work, we plan to estimate the criticality of a component based on its location and connectivity in an architecture. This will help to identify the components that are critical to system safety.

References

1. A. Avizienis, J.C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing", *IEEE Transactions on Dependable and Secure Computing*, Mar 2004, Vol: 1, pp. 11- 33.
2. L. Bass, P. Clements, and R. Kazman, "Software Architecture in Practice". 2-*nd* eddition. 2003: Addison-Wesley.
3. J. Hill and D. Victor, "The Product Engineering Class in the Software Safety Risk Taxonomy for Building Safety-Critical Systems", *Proc. of the 19th Australian Conference on Software Engineering*, 2008, pp. 617-626.
4. N.G. Leveson, "Software safety: why, what, and how", *ACM Computing Surveys (CSUR) archive*, Jun 1986, Vol 18, pp. 125-163.
5. N.G. Leveson and P.R. Harvey, "Analyzing Software Safety", *IEEE Trans. on Software Engineering*, Sep 1983, Vol SE-9, NO. 5, pp. 569-579.
6. B. Littlewood and L. Strigini, "Software reliability and dependability: a roadmap", *Proc. of the 22nd IEEE International Conference on Software Engineering on the Future of Software Engineering (ICSE'00)*, Limerick, Ireland, 2000, pp. 175-188.
7. A. Mohamed and M. Zulkernine, "Improving Reliability and Safety by Trading off Software Failure Criticalities", *Proc. of the 10th IEEE International Symposium on High Assurance System Engineering*. Nov 2007, Dallas, Texas, pp. 267-274.
8. A. Mohamed and M. Zulkernine, "On Failure Propagation in Component-Based Software Systems", *Proceedings of the 8th IEEE International Conference on Quality Software*, IEEE CS Press, Oxford, UK, 2008, Pg: 402-411.
9. Object Management Group, "OMG Unified Modeling Language (OMG UML)", Superstructure, Version 2.1.2, *OMG Available Specification without Change Bars*, formal/2007-02-05, Nov 2007.
10. M.B. Swarup and P.S. Ramaiah, "An Approach To Modeling Software Safety", *Proc. of the 9th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, 2008, pp. 800-806.
11. C. Szyperski, "Component software: beyond object-oriented programming", *Addison-Wesley*, 1998, ISBN 0-201-17888-5.
12. A.C. Tribble and S.P. Miller, "Software Intensive Systems Safety Analysis", *IEEE A&E Systems Magazine*, Oct 2004, pp. 21-26.
13. W. Weihang and T. Kelly, "Safety tactics for software architecture design", *Proceedings of the 28th Annual International Conference on Computer Software and Applications.*, York Univ., UK, Sep 2004, pp. 368-375.