2009

Progress of Computer Technology and Application 2009'

计算机技术与应用进展

《计算机技术与应用进展》编委会 编

上册

中国科学技术大学出版社

Progress of Computer Technology and Application in 2009

计算机技术与应用进展

· 2009 ·

中国仪器仪表学会

中国系统仿真学会

中国仪器仪表学会
微型计算机应用分会

教育部 安全关键
工业测控技术工程研究中心

合肥工大高科

广西大学

中国科学技术大学出版社

2009 · 合肥

图书在版编目(CIP)数据

计算机技术与应用进展·2009/刘晓平，等主编. —合肥：中国科学技术大学出版社，2009.7
ISBN 978-7-312-02565-5

Ⅰ. 计… Ⅱ. ①刘… ②蒋… ③李… Ⅲ. ①计算机科学—文集 ②计算机应用—文集 Ⅳ. TP3-53

中国版本图书馆 CIP 数据核字（2009）第 096170 号

书　　名：计算机技术与应用进展·2009
著作责任者：刘晓平、蒋建国、李琳
责 任 编 辑：张善金
出　版　者：中国科学技术大学出版社
地　　址：合肥市金寨路 96 号　　邮编：230026
网　　址：http//www.press.ustc.edu.cn
电　　话：发行部 0551-3602905　　邮购部 3602906　　编辑部 3602910
电 子 信 箱：press@ustc.edu.cn　edit@ustc.edu.cn
印　刷　者：合肥学苑印务有限公司
发　行　者：中国科学技术大学出版社
经　销　者：全国新华书店
开　　本：880mm×1230mm　1/16　印张：72　字数：2348 千
版　　次：2009 年 7 月第 1 版　　2009 年 7 月第 1 次印刷
定　　价：298.00 元

Progress of Computer Technology and Application in 2009

全国第 20 届计算机技术与应用（CACIS）学术会议

大会主席：　　韩江洪　张协奎

程序委员会主席：刘晓平　钟　诚

组织委员会主席：李陶深　张建军

大会程序委员：（排名不分先后）

曹广忠　曹　军　陈家新　陈军宁　陈　明　程　恩　程仁洪　冯冬青

龚　昕　韩江洪　韩晓微　胡成全　纪秀花　贾根莲　简　炜　蒋建国

刘晓平　潘汉达　秦　锋　热合木江　邵晨曦　宋宜斌　宋执环　王　军

王晓峰　王忠群　吴乐南　武　文　徐汀荣　尹建华　郁　滨　余成波

袁　涛　查红彬　张继福　张　琳　钟　诚　梁华国　罗月童　李　琳

郑利平　路　强　徐本柱　石　慧

主编：刘晓平　蒋建国　李　琳

主审：韩江洪　吴乐南　钟　诚　邵晨曦　宋执环　王晓峰

前　言

中国仪器仪表学会和中国系统仿真学会长期致力于计算机科学与技术的研究与应用推广工作， CACIS 工作年会已成为全国信息学科相关专业互相渗透和交流的重要平台。全国第20届计算机技术与应用学术会议（CACIS·2009）暨全国第1届安全关键技术与应用学术会议将于2009年7月17-22日在广西大学举行。本届会议将聚集国内外知名专家学者，交流信息理论与应用的研究成果，探讨计算机技术应用、建模仿真以及安全关键技术中的挑战性问题。

主办单位：中国仪器仪表学会（CIS），中国系统仿真学会（CSSS），中国仪器仪表学会微型计算机应用学会（CACIS），中国系统仿真学会复杂系统建模与仿真计算专业委员会筹备处（CSSC）

承办单位：合肥工业大学、教育部安全关键工业测控技术工程研究中心

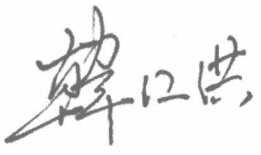
协办单位：广西大学计算机与电子信息学院、广西大学科技处、广西玉林师范学院、合肥工大高科信息技术有限责任公司、《仪器仪表学报》、《计算机辅助设计与图形学学报》、《系统仿真学报》、《工程图学学报》

会议地点：广西南宁，广西大学

本次学术会议的重点主题是“安全关键技术及其应用”，尤其是系统安全与工业现场安全的相关问题，会议筹备委员会自2008年12月发出第一轮征文通知后，共收到学术论文416篇，内容涵盖了安全关键技术及其应用、计算机仿真方法与应用、计算机辅助设计与图形学、人工智能与算法、软件工程与软件设计、数据库与信息系统、仪器仪表与检测控制、图像与多媒体技术、网络与通讯等相关主题，具有广泛的代表性，经大会程序委员会通讯评审和集中复审，确定了217篇收入由中国科学技术大学出版社正式出版的会议论文集，并初步评选出优秀论文65篇，大会宣读后的优秀论文将于会后分别被推荐至《仪器仪表学报》、《计算机辅助设计与图形学学报》、《系统仿真学报》、《工程图学学报》、《合肥工业大学学报》、《广西大学学报》。

特别鸣谢：中国科学院沈绪榜院士、奥地利嵌入式 LINUX 专家 Nicholas Mc Guire 教授、台北教育大学 Timothy K. Shih 教授、工大高科信息技术有限责任公司魏臻总裁。

7月的广西期待着您的到来！



中国仪器仪表学会微型计算机应用学会理事长

广西大学副校长

2009年6月11日

目　次

大会报告

上　册

安全关键应用

网络与通讯

人工智能与算法

图像与多媒体

下 册

计算机辅助设计与图形学

仿真理论及实践

数据库与系统

软件工程

仪器仪表

Floss Software Architecture for Safety Related Systems

Nicholas Mc Guire

Distributed and Embedded Systems Lab, Lanzhou University

Abstract: The utilization of COTS/FLOSS in the context of safety related systems is of interest for technical as well as economical reasons and there are on-going efforts in industry to find solutions to the challenging question of how to utilize FLOSS in safety applications. Substantial research efforts have reached positive conclusions about utilizing COTS, albeit with restrictions [2],[5],[4].

While building on high-complexity software is in principle contended in much of the safety community, we believe that the vast resource of FLOSS components is of such great utility that it is mandatory to investigate there usage in safety related systems and find answers to the open questions.

Safety critical system design been focused on minimizing complexity of the system most notably of software components which is also suggested by key standards [61508-37.4.2.6]. Here we will argue that while there are grounds to claim that GNU/Linux is conservative with respect to design and technology, it is obviously large and complex, never the less usable if an appropriate solution at the architectural level is designed. These solutions need to address a deeper problem of the safety community, with respect to software: how can we address high software complexity - it is not going to g down in the future and answers need to be found based on sound technical reasoning.

1. Introduction

Utilization of COTS/FLOSS in the context of safety related systems is of interest for technical and also economical reasons. While it is highly contended to utilize high complexity software for safety related systems all together. There have been successful demonstrations of such use at the highest safety integrity level (i.e. SICAS [3], EN 50128, based on diversity).

We believe that the vast resource of FLOSS components is of such great utility that it is mandatory to investigate there usage in safety related systems. Aside from the economical aspects, security aspects, long term stability and reliability play a decisive role why industry is looking into the use of COTS/FLOSS based systems for safety applications.

While space limitations prohibit an extensive design discussion, we outline a possible basic software architecture, allowing to utilize FLOSS for safety related systems.

2. FLOSS for safety related systems

Safety critical systems have traditionally been focused on minimizing complexity of the system - most notably of software components. This is not only due to the exponential efforts of assessment and validation but also firmly rooted in the guiding standards [61508-3 7.4.2.6]. This seems to almost preclude the usage of FLOSS components

in safety related systems (FLOSS being one form of COTS software [prEN 50128 3.18]). Generally 61508 and its derived standards reiterate frequently that

- size
- complexity
- novelty of design
- novelty of technology

are considered critical factors to consider when selecting technical solutions. In this presentation we will argue that while there are grounds to claim that GNU/Linux is conservative with respect to design and technology, it is obviously large and complex, never the less usable if an appropriate solution at the architectural level is designed. These solutions need to address a deeper problem of the safety community, with respect to software: how can we manage the very high complexity of FLOSS based systems ?

Why COTS for safety related systems ?

Before looking at how COTS/FLOSS could be utilized we should look at why this is being discussed in the first place. Taking clause 7.4.2.6 of IEC 61508 which is echoed in many of its derived application specific standards, one might reject the concept of FLOSS for safety related systems in principle - some reasons why we think it is worth looking at:

- recent safety standards are amended with stringent security requirements – one area where FLOSS has proven its capabilities and many bespoke safety critical operating systems have deficits.
- Flexibility of design, most notably of the non-safety related components in the system used for monitoring, remote maintenance or auxiliary services packed on the same hardware simply because the platforms are becoming more and more potent.
- The plethora of available components in the FLOSS community is large and rapidly growing, mandating that an answer in the context of safety related systems for this economically attractive pool of technology be found.
- Stability of COTS/FLOSS systems - though this might well be contended, the stability (reliability,availability) of COTS/FLOSS based systems has proven to be high - most notably in high-complexity systems - it is not a coincidence that quite a few of the top 500 clusters are based on GNU/Linux (notably the current Nr1:Roadrunner)
- Industry is trying to consolidate its OS usage, with GNU/Linux being used widely in embedded industrial applications, extending its use into the high-cost area of safety related systems seems economically sensible.
- Portability properties of FLOSS components
- Maintainability issues, notably personnel availability and long term stability of interfaces (i.e. POSIX, Single UNIX Specifications [?]).

This list of arguments is by no means complete and it is not claiming that usage of opensource in safety related systems is the solution to all safety problems, but it is worth looking into FLOSS as an enhancement of options, with some technical and economic properties that are not found in commercial competitors.

In our research work at DSLab we have identified 6 general approaches to utilizing FLOSS for safety related systems [6], we are not able to outline all of them here, butwill focus on one of them - Application level safety.

3. A sample FLOSS safety architecture for application level safety

Traditional approaches of KISS assume that the role of software can be reduced to a minimum, that we can actually know each detail, know the behavior of the system based on rigorous design methodologies, given a specific input data set and a well defined environment. But with large open-source components the picture changes,

we no longer are able to constraint the usage to a well known set of meticulously designed functions, nor do we know the details of how the system will behave on particular inputs - what we do have on the other hand is a level of information on field service that clearly exceeds the information typically available for bespoke software components.

One approach that might come to mind now would be to focus on proven-in-use arguments. While this might be a suitable strategy for some systems, notably those in the SIL1/2 range, we don't see proven-in-use as a sufficient argument on its own – even with extensive, and high-quality bug-tracking available for key GNU/Linux components.

While this data can provide valuable input for selecting components and will impact the ease of certification we propose to investigate architectural options to ensure the required level of safety integrity. Building on diversity i.e. as IEC 61508-6 Appendix E suggests, is one option to utilize COTS software components in safety critical systems, here we would like to introduce an alternative approach based on placing the COTS hardware and software components into a "gray-channel" arguing the SIL at the application level only.

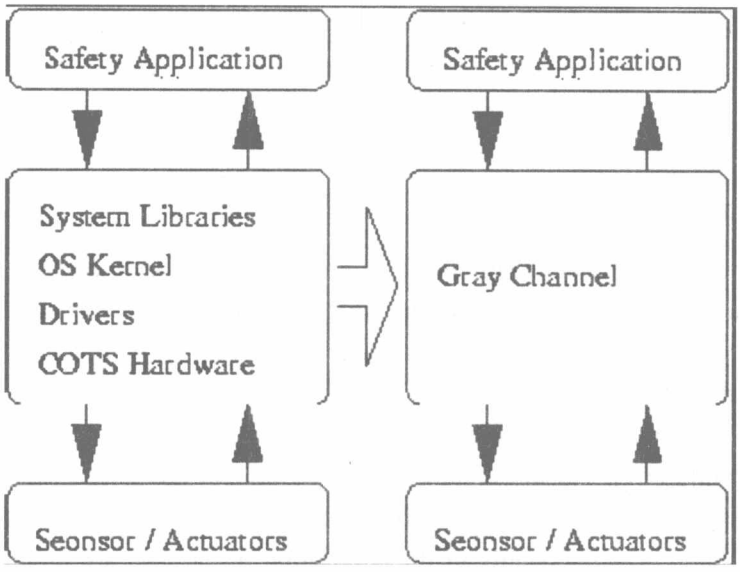


Figure1: Gray Channel for COTS/FLOSS

The application model is a general "control-loop" of read data, process data, write data with the data coming from the sensors and results going to the actuators. This model is not concerned with the correctness of the bespoke algorithm nor with the implementation of the same. The safety application is assumed to be a bespoke software components, achieving its safety integrity by a suitable process - it is assumed to be suitable for the SIL in question. The Sensor/Actuator is considered to be safe appropriate to the required SIL and also no considered, further, and this is still a limitation of this model, we are assuming that there is no direct feedback possibility, at the software level, from thegray-channel to the sensor/actuator other than the data channels (i.e. via Ethernets).

4. Fault Class Mapping

Instead of a traditional FMEA type mapping we approach the problem slightly differently- we look at general fault classes and which of them could lead to detection problem and thus impact the safety properties of the safety application.

To argue such a container of "unknown software", potentially allowing to utilize COTS components of arbitrary

complexity we use a generic fault model.

- Random Faults that impact the properties of the gray-channel
- Systematic Faults that impact the properties of the gray-channel
- Random Faults in the gray-channel that "feed-back" to the safety application
- Systematic Faults in the gray-channel that "feed-back" to the safety application

If all four can be shown to lead to no undetected fault at the safety application, we believe this is a potentially suitable architecture to utilize COTS components of high-complexity - i.e. GNU/Linux.

4.1. Faults within the gray-channel

For data traversing the gray-channel EN 50159 has suitable methods available, some of the needs resulting are

- appropriate checksums on data objects - detection of random and systematic modification in the data channels.
- sequence numbers on data objects being transmitted, ensuring the detection of retransmission of old data or missing of a data object.
- timeout response on failure to receive data at the receiving side, ensuring that a fail-silent application does not go undetected.

The problems of random and systematic faults within a communication channel, which is a full featured FLOSS OS in this case, can be considered resolved.

4.2. fault feed-back to the safety application

The more problematic realm is the faults introduced by unintended feedback from the gray-channel to the safety related software. This can be split into the two brad categories of random and systematic faults.

4.2.1. Random faults

To address random faults, the simplest option is to provide redundancy at the application level, note that we are assuming replication of the identical application, with the constraint that there are no functionally shared components at the application level. The constraints introduced are that the sensor/actuator must be able to handle multiplexing demultiplexing and voting of data received from the duplicated (but independent) applications, and thus random faults impacting the functionality of the application or the data are mitigated at the sensor/actuator level.

4.2.2. Systematic Faults

The big problem are the systematic faults that the components in the gray-channel may inflict on the safety application, OS and hardware have the quite obvious potential to directly impact any application level flow of control or data objects. To mitigate systematic faults we distinguish between text segment (code) manipulation and data segment manipulation.

Text segment modification:

Text segment modification can be detected by traditional means of running CRC checks on the text-segments, this could be run at application level on a per-request basis preceding any calculation, to ensure that the check it self is not disabled by faults in the text-segment, qualification of the test-results are left to the actuator, thus the property of text-segment correctness is diversely represented and manipulation can be detected again at the (external) actuator level.

Data modifications:

Systematic data modification are problematic because the constitute single points of failure, a single instruction i.e. adding two values modified to subtract the same two values by sign inversion of one of the values (a single bit

is flipped) would not alter the flow of execution nor would it necessarily break the algorithmic logic - thus stay undetected.

A systematic fault has the property of introducing a unintended state transition in a system when a particular context is given - every time this context is presented to the system. Thus one defense would be diversity at the application level. Coded monoprocessors have been proposed to mitigate these faults, explicit N-version programming (with all its known problems) has been in discussion, and forms of automated or inherent diversity also look like an option to be looked into.

With these constraint in mind we arrive at a possible architecture allowing to build on complex COTS software components that looks like this:

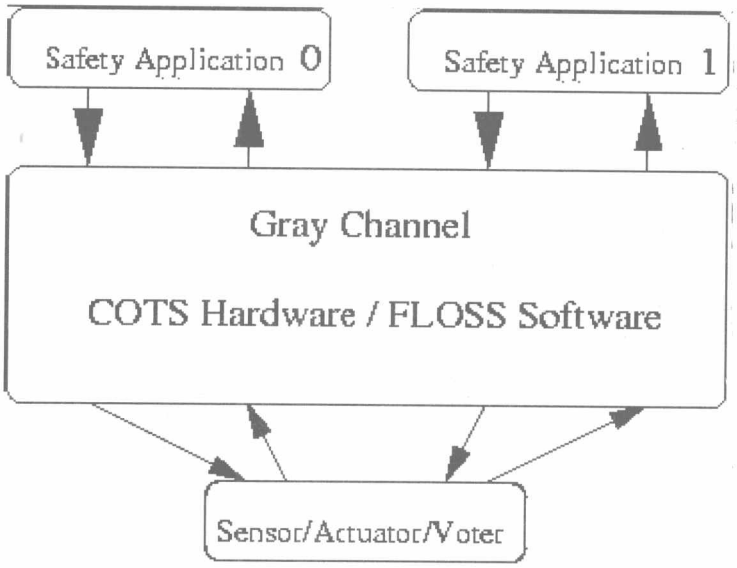


Figure2: Application Level Safety

5. Conclusion

With appropriate software architecture utilization of COTS/FLOSS for safety critical application is feasible. We believe that for technical and economical reasons the option to utilize FLOSS in this challenging domain is worth the effort of resolving open issues with integration of unsafe software components of high complexity for systems of even the highest safety integrity levels. Safety standards have been moving forward on the issue of FLOSS usage (notably prEN 50128) and we expect more functional safety standards to address the open-issues in the future. Usage of FLOSS is not the answer to all safety problems, and such a paradigm change, in it self, exhibits a potential to introduce hazards, thus it needs to be done carefully and based on an open discussion in the safety community. We hope that we can contribute to igniting this discussion.

计算机的技术演变

沈绪榜

中国航天电子基础技术研究院研究员

“每一次长周期的技术大革命的爆发和大规模扩散都要引发整个经济社会的结构性转变。”上世纪美国最后一位诺贝尔奖获得者发明的集成电路，促进了硅基芯片的长周期的技术大革命，特别是微处理器的发明，实现了计算机的廉价制造和批量生产。促进了计算机产业与电信产业的发展，使9亿人口进入了知识社会。技术可以造就许多种未来，我们真正得到的未来取决于技术和其它因素的相互作用。这些因素包括法律、规章、文化，特别是人。“谁能适应不断加快的技术变革，谁就能生存下去。”计算机的技术演变包括设计、应用和制造三个方面。

研究表明：由于日益增长的晶体管数量和越来越快的晶体管开关速度，集成电路的能力正以每年74%的速度增长。而常规体系结构的处理器尽管采用了深度流水线技术，试图支持每个周期能执行一条指令，但预计计算机的性能每年只有19%的增长速度。这些数字差别表明，计算机的设计具有极大的发展机会，它的设计技术演变包括算法、模式与结构的演变。

计算机应用的技术演变是以芯片的技术演变为基础的，有高性能计算，网络计算，嵌入式计算，以及飞天梦想与人工智能。计算技术的太空应用是因为“将地球、空气和重力抛在身后，是人类自古以来的梦想（2003年2月“哥伦比亚”号7人遇难）”。人工智能（Artificial intelligence)一词是在1956年Dartmouth学会上提出的，其目的就是要让机器人能够像人一样思考。

现代计算机实现技术进步的基础，是贝尔实验室的固体物理学家肖克莱（W.Shockley,1910-1989）和巴丁（J.Bardeen,1908-1991）以及布莱顿（W.H.Brattain,1902-1987），于1947年12月23日发明的晶体管，1956年获得诺贝尔物理学奖，晶体管使计算机的体积、功能、速度、价格与可靠性等方面取得了划时代的进步。1958年杰尔·基尔比（Kilby）研制成功了世界上的第一块晶体管集成电路（IC, Integrated Circuit）,又叫做芯片（Chip，die）。相隔42年之后的2000年，基尔比获得了诺贝尔物理学奖，人们肯定了芯片在信息社会发展中的基础作用。硅基芯片是按照摩尔预言的速度演变的。

上世纪50年代IBM曾完成过一次市场调研，此调研预言全世界只需要6台计算机；1978年IBM的数字设备公司的执行总裁Ken OLSON说，他不能够想象为什么每一个人需要一台计算机；20世纪80年代中期，Biu Gates说，没有人需要大于640K内存的计算机 这些非常不准确的预测表明，计算机应用在技术方面所积累的量变，最终会导致质的改变，正像摩尔定律所预言的那样。目前的问题不是技术演变的路标，而是芯片制造的产业现实，如今开工一家新的工厂需要投入30亿美元。真正的突破并非是技术演示，而是制成人们可以使用的系统。

2009年6月