S-ADA: Software as an Autonomous, Dependable and Affordable System

Kai-Yuan Cai[*] [1]
[*] *Department of Automatic Control*
Beijing University of Aeronautics
and Astronautics, Beijing 100191, China
kycai@ buaa.edu.cn

Kishor S. Trivedi[※]
[※] *Department of Electrical and*
Computer Engineering, Duke University
Durham, NC 2770, USA
kst@ee.duke.edu

1. Introduction

ADA is a popular programming language that was named after Lady Ada Lovelace (1815-1852) and recommended by Department of Defense, USA, for development of large scale safety-critical software systems. In this Fast Abstract, ADA is reinterpreted as Autonomous, Dependable and Affordable. It is argued that future software systems should be ADA. A few affordable techniques are briefly described to highlight how to make software systems ADA.

2. Driving Forces for a New Vision

The last two decades have witnessed tremendous changes in the vision of software. This was driven by several closely related research endeavors. First, in October 2001, IBM released a manifesto observing that the main obstacle to further progress in the IT industry was a looming software complexity crisis, and proposed the idea of autonomic computing as a response to the crisis [1]. An autonomic computing system can manage itself given high-level objectives from administrators and has capability of self-configuration, self-optimization, self-healing, and self-protection.

Second, software agents have received considerable amount of research attention in software engineering community since the first special issue of the Communications of the ACM on agents that appeared in 1994 [2]. Software agents are different from ordinary programs, objects and expert systems, in that they have persistence (code is not executed on demand but runs continuously and decides for itself when it should perform some activity), autonomy (agents have capabilities of task selection, prioritization, goal-directed behavior, decision-making without human intervention), social ability (agents are able to engage other components through some sort of communication and coordination, they may collaborate on a task) and reactivity (agents perceive the context in which they operate and react to it appropriately) [3]. Software agents should be adaptive.

Third, cyber-physical systems [4] and Internet of things [5] have given new research thrusts to academia, industry and government. A cyber-physical system such as air transportation system is a network of embedded systems that bridges the cyber-world of computing, communication and control with the physical world. On the other hand, the Internet of Things refers to uniquely identifiable objects (Things) and their virtual representations in an Internet-like structure. It will be able encode 50 to 100 trillion objects, and be able to follow the movement of these objects.

Fourth, in June 2006 the Software Engineering Institute of Carnegie Mellon University released a report and argued that the sheer scale of Ultra-Large-Scale (ULS) systems would change everything [6]. For example, people will not just be users of a ULS system; they will be elements of the system. ULS systems are not just systems or systems of systems, but are socio-technical ecosystems. The report also identifies design and evolution, orchestration and control, and monitoring and assessment as fundamental challenges of ULS systems we need to face.

Finally, cloud computing has become a mainstream research direction in computing community [7]. Cloud computing refers to the logical computational resources (data, software) accessible via a computer network (through WAN or Internet etc.), rather than from a local computer. Software is seen as a service, and cost is a major criterion for cloud computing to succeed.

Obviously, software is an integral part of social infrastructure and daily life. Then, a natural question arises: what requirements should be placed on software that will meet the needs of autonomic computing, software agents, cyber-physical systems, Internet of things, ultra-large-scale systems, and cloud computing? Are we ready to have a new vision of software requirements?

[1] Cai was supported by the National Natural Science Foundation of China (Grant Number 60973006) and the Beijing Natural Science Foundation (Grant Number 4112033).

3. Software as an ADA System

Given various great research endeavors and driving forces identified in the last section, it is reasonable to argue that we should have a new vision of software: future software systems need to meet three-dimensional requirements: autonomy, dependability, and affordability. That is, software should serve as an Autonomous, Dependable and Affordable (ADA) System.

By being autonomous we mean being autonomic and being intelligent. In order to meet the requirements of autonomic computing, future software systems need to manage themselves. However self-management should be achieved in a cost-effective and intelligent manner. Intelligence implies learning, reasoning and adaptation. Autonomic software systems should also be intelligent enough to make appropriate decisions and responses in various expected or unexpected environmental scenarios as required by adaptive software and software agents.

By being dependable we mean that the chance of software systems suffering from the negative consequences of various bugs should be acceptably small. This implies software systems should be safe, highly reliable, highly available, highly secure, and so on. How to make software systems dependable is still a challenge to software community. Statistics shows a growing trend of software errors in computing systems, and evidence does not suggest that this trend is overridden by the research endeavors identified in the last section [8].

By being affordable we mean that both software systems and software technologies being affordable. Since software has become an integral part of social infrastructure and daily life for ordinary people, it should be as cheap as water and power supply. Without affordable software systems and technologies, people would not be able to use cyber-physical systems, Internet of things, and ULS systems. The dream of cloud computing would not become a reality. Software as a service should not be confined to the rich.

4. How to Make Software Systems ADA

In this section we identify a few affordable techniques that may help make future software systems ADA.

Software Rejuvenation. Software shows aging behavior akin to hardware aging due to aging-related bugs such as memory leaks. Software aging phenomenon refers to performance degradation and increased failure rate even if software code is not modified. A simple proactive way to counter software aging is to rejuvenate software at appropriate times before failures occur [9].

Software Fault-Tolerance by Environmental Diversity. A bug can cause software to exhibit a chaotic and even nondeterministic behavior with respect to the occurrence and nonoccurrence of failures if its activation process is complex, depending on the system-internal environment. These bugs are referred to as Mandelbugs. In comparison with the expensive techniques of software fault-tolerance based on design diversity, an affordable technique of software fault-tolerance is to carry out a retry or failover to a replica, based on the observation that the system-internal environment will be different and hence diverse for the next software execution [10].

Software as a Control System (SaaCS). Following the idea of software cybernetics, a reasonable way to make software autonomous is to treat software as a (feedback, adaptive) control system, which comprises at least two components: controlled object and controller [11]. An essential feature here is that both the components are software in themselves, and control is applied to the software behavior to guarantee its dependability and quality of service in an autonomous manner.

Dynamic Random Testing (DRT). Random testing is a popular technique for software quality assurance. A simple technique that can improve the effectiveness of random testing with little overhead is DRT [12]. Suppose that the input domain of the software under test comprises a number of subdomains. Test cases are selected from these subdomains with certain probabilities. DRT dynamically updates these probabilities, depending on if test cases selected from the corresponding subdomains reveal failures.

5. Concluding Remarks

Given the new vision that software should be treated as an ADA system and as a control system, a natural further step beyond S-ADA is to formulate new and holistic paradigms: ADA computing and ADA control.

References

[1]. J.O.Kephart, D.M.Chess, "The Vision of Autonomic Computing", *Computer*, January 2003, pp41-50.

[2]. P.Maes, "Agents that Reduce Work and Information Overload", *CACM*, Vol.37, No.7, 1994, pp31-40.

[3]. http://en.wikipedia.org/wiki/Software_agent#History

[4]. R.Rajkumar, I.Lee, L.Sha, J.Stankovic, "Cyber-Physical Systems: The Next Computing Revolution", *Proc. DAC*, 2010.

[5]. http://en.wikipedia.org/wiki/Internet_of_Things

[6]. SEI, "Ultra-Large-Scale Systems: The Software Challenge of the Future" Carnegie Mellon University, June 2006.

[7]. M.Armbrust, et al, "Above the Clouds: A Berkeley View of Cloud Computing", UC at Berkeley, February 2009.

[8]. M.Grottke, A.P.Nikora, K.S.Trivedi, "An Empirical Investigation of Fault Types in Space Mission System Software", *Proc. DSN*, 2010.

[9]. Y.Bao, X.Sun, K.S.Trivedi, "Adaptive Software Rejuvenation: Degradation Model and Rejuvenation", *Proc. DSN*, 2003.

[10]. M.Grottke, K.S.Trivedi, "Fighting Bugs: Remove, Retry, Replicate, and Rejuvenate", *Computer*, February 2007.

[11]. K.Y.Cai, J.W.Cangussu, R.A.DeCarlo, A.P.Mathur, "An Overview of Software Cybernetics", *Proc. STEP*, 2003, IEEE, pp77-86.

[12]. K.Y.Cai, H.Hu, C.H.Jiang, F.Ye, "Random Testing with Dynamically Updated Test Profile", *Proc. ISSRE* 2009, Fast Abstract 198.