Persistent Computing: What is it and Why Study it?

Jingde Cheng

Department of Information and Computer Sciences
Saitama University, Saitama, 338-8570, Japan
cheng@ics.saitama-u.ac.jp

Abstract

The requirement that a computing system should run continuously and persistently is never taken into account as an essential and/or general requirement by traditional system design and development methodologies. A traditional computing system often has to stop its running and service when it has some trouble, it is attacked, and it needs to be maintained, upgraded, or reconfigured. This paper presents the author's vision of persistent computing, a new methodology and/or paradigm that aims to develop continuously dependable and dynamically adaptive reactive-systems, called "Persistent Computing Systems," in order to build more human-friendly reactive systems. The paper presents the author's considerations on why we should study persistent computing systems, discusses how persistent computing systems can be constructed to satisfy their requirements, and shows some new scientific and technical challenges on persistent computing.

1. Introduction

The requirement that a computing system should run continuously and persistently is never taken into account as an essential and/or general requirement by traditional system design and development methodologies. A fact to support this proposition is that we cannot find 'persistence' and/or 'persistent' related technical terms defined or listed in various computer dictionaries, computer glossaries, encyclopedia of software engineering, and handbooks of software reliability such as [22, 24, 25, 27, 28, 30, 36]. Although there are some individual computing systems designed and developed with considerations on fault tolerance [1, 21], in general a traditional computing system often has to stop its running and service when it has some trouble, it is attacked, and it needs to be maintained, upgraded, or reconfigured.

However, modern society is more and more dependent on various computing systems, and therefore, dependent on the continuous, reliable, and secure functioning of the systems. On the other hand, some research area, such as autonomous evolution, agent society, anticipatory systems, and artificial life, also require continuous, reliable, and secure functioning of computing systems in order to simulate some life and/or social phenomena.

Can we make computing systems continuously live for functioning persistently? This paper presents the author's vision of persistent computing, a new methodology and/or paradigm that aims to develop continuously dependable and dynamically adaptive reactive-systems, called "***Persistent Computing Systems***," in order to build more human-friendly reactive systems. The paper presents the author's considerations on why we should study persistent computing systems, discusses how persistent computing systems can be constructed to satisfy their requirements, and shows some new scientific and technical challenges on persistent computing.

2. Why Persistent Computing Systems?

A reactive system is a computing system that maintains an ongoing interaction with its environment, as opposed to computing some final value on termination [20, 29]. A ***persistent computing system*** is a reactive system that functions continuously and evolves autonomously anytime without stopping its reactions even when it had some trouble, it is being attacked, or it is being maintained, upgraded, or reconfigured. Note that if a computing system to compute some final value on termination or to completely stop its computation due to some reason, then in general it is not a persistent computing system.

The first problem motivated the present author to study persistent computing systems and/or persistent computing is to solve the problem of automated theorem finding [7, 8]. Wos in 1988 proposed 33 basic research problems in automated reasoning [34, 35]. The thirty-first one is the problem of automated theorem finding (ATF for short): "What properties can be identified to permit an automated reasoning program to find new and interesting theorems, as opposed to proving conjectured theorems?" The problem of ATF is still completely open until now. The most important and difficult requirement of the problem is that, in contrast to proving conjectured theorems supplied by the user, it asks properties and/or criteria such that an automated reasoning program can use them to find some theorems in a field that must be evaluated by theorists of the field as new and interesting theorems. The significance of solving the problem is obvious because an automated reasoning program satisfying the requirement can provide great assistance for scientists in various fields. In order to find new theorems in a special field, we have to tell an ATF system what are known concepts, theorems, open problems in the field at first, and then run the system continuously. Because an ATF process needs a lot of computation power and time, and produces a lot of intermediates, any interruption of the process is very undesirable. This naturally led us to requiring persistent computing.

The second problem motivated the present author to study persistent computing systems and/or persistent computing is to develop autonomously and continuously evolutionary systems [11, 17]. The term 'evolution' means a gradual process in which something changes into a different and usually better, maturer, or more complete form. The autonomous evolution of a system, which may be either natural or artificial, should be a gradual process in which everything changes by conforming to the system's own laws only, and not subject to some higher ones. Because any evolution is a gradual process and in general the outside environment of a system changes over time, the autonomous evolution of a system should be a persistently continuous process without stop of interactions with its outside environment. Stopping a system means that its evolution is interrupted. Although a stopped system may be resumed, it may be stranded since its outside environment has changed. Therefore, an autonomously evolutionary system should at the same time be a continuously evolutionary system. This requires persistent computing.

The third problem motivated the present author to study persistent computing systems and/or persistent computing is anticipatory computing, in particular, anticipatory reasoning-reacting systems. The concept of an anticipatory system first proposed by Rosen in 1980s [6, 31]. Rosen considered that "an anticipatory system is one in which present change of state depends upon future circumstance, rather than merely on the present or past" and defined an anticipatory system as "a system containing a predictive model of itself and/or its environment, which allows it to change state at an instant in accord with the model's prediction to a latter instant." An anticipatory reasoning-reacting system (ARRS for short) is a computing system containing a controller C with capabilities to measure and monitor the behavior of the whole system, a traditional reactive system RS, a predictive model PM of RS and its external environment, and an anticipatory reasoning engine ARE such that according to predictions by ARE based on PM, C can order and control RS to carry out some operations with a high priority. It is the predictions by ARE based on PM that makes an ARRS able to take anticipation [12, 13]. All anticipatory systems have the following two characteristics in common: (1) for any anticipatory system, concerning its current state, there must be a future state referred by the current state, and (2) for any anticipatory system, its states form an infinite sequence. Therefore, we can say that the notion of anticipatory system implies a fundamental assumption or requirement, i.e., to be anticipatory, a commuting system must behave continuously and persistently without stopping its running. It is obvious that any anticipatory computing and/or reasoning process should not be interrupted. Thus, anticipatory reasoning-reacting systems and anticipatory computing also requires persistent computing.

The fourth problem motivated the present author to study persistent computing systems and/or persistent computing is ubiquitous computing. The ultimate goal of ubiquitous computing is to provide users with the way of computing anytime and anywhere [33]. Obviously, a necessary condition and/or fundamental assumption to underlie ubiquitous computing are that there certainly are systems functioning anytime available throughout the physical world. Therefore, ubiquitous computing must lead to requiring that computing systems function continuously and persistently, i.e., persistent computing [15].

From different viewpoints, persistent computing systems and/or persistent computing may provide us with different benefits. From the viewpoints of computational science and engineering, persistent computing systems can provide us with continuous computing powers which we need to establish the computational methodology as a third paradigm of scientific methodology. From the viewpoints of reliability and security, persistent computing systems can serve as infrastructures for achieving high reliability and high security in the real world. From the viewpoint of autonomous evolution, agent society, anticipatory systems, and artificial life, persistent computing can be considered as the purpose rather than a way or tool.

3. Design and Development of Persistent Computing Systems

The present author has proposed the following general principles in concurrent systems engineering [9, 10]:

The ***wholeness principle of concurrent systems***: "The behavior of a concurrent system is not simply the mechanical putting together of its parts that act concurrently but a whole such that one cannot find some way to resolve it into parts mechanically and then simply compose the sum of its parts as the same as its original behavior."

The ***uncertainty principle in measuring and monitoring concurrent systems***: "The behavior of an observer such as a run-time measurer or monitor cannot be separated from what is being observed."

The ***self-measurement principle in designing, developing, and maintaining concurrent systems***: "A large-scale, long-lived, and highly reliable concurrent system should be constructed by some function components and some (maybe only one) permanent self-measuring components that act concurrently with the function components, measure and monitor the system itself according to some requirements, and pass run-time information about the system's behavior to the outside world of the system."

On the other hand, we can say the following ***dependence principle in measuring, monitoring, and controlling***: "A system cannot control what it cannot monitor, and the system cannot monitor what it cannot measure."

Based on the above principles and the fact that system buses have successfully played a very important role in hardware reconfiguration and upgradability in computer engineering, the present author considered that a persistent computing system can be constructed by a group of control components including self-measuring, self-monitoring, and self-controlling components with general-purpose which are independent of systems, a group of functional components to carry out special takes of the system, some data/instruction buffers, and some data/instruction buses. The buses are used for connecting all components and buffers such that all data/instructions are sent to target components or buffers only through the buses and there is no direct interaction which does not invoke the buses between any two components and buffers.

Conceptually, a ***soft system bus***, SSB for short, is simply a communication channel with the facilities of data/instruction transmission and preservation to connect components in a component-based system [15]. It may consist of some ***data-instruction stations***, which have the facility of data/instruction preservation, connected sequentially by ***transmission channels***, both of which are implemented in software techniques, such that over the channels data/instructions can flow among data-instruction stations, and a component tapping to a data-instruction station can send data/instructions to and receive data/instructions from the data-instruction station.

An ***SSB-based system*** is a component-based system consisting a group of control components including self-measuring, self-monitoring, and self-controlling components with general-purpose which are independent of systems, and a group of functional components to carry out special takes of the system such that all components are connected by one or more SSBs and there is no direct interaction which does not invoke the SSBs between any two components [15].

The most intrinsic characteristic or most important requirement of SSBs is that an SSB must provide the facility of data/instruction preservation such that when a component in a system cannot work well temporarily all data/instructions sent to the component should be preserved in some data-instruction station(s) until the component works well to get these data/instructions. Therefore, other components in the system should work continuously without interruption, except those components that waiting for receiving new data/instructions sent from the component in question.

From the viewpoint of structure, an SSB may be either linear or circular. On the other hand, from the viewpoint of information flow direction, data/instruction flows along an SSB may be either one-way or bidirectional. Therefore, there may be four types of SSBs: ***linear one-way***, ***linear bidirectional***, ***circular one-way***, and ***circular bidirectional*** SSBs. It is obvious that different types of SSBs will provide system designers and developers a variety of technical benefits and functional advantages to make target systems more flexible and powerful. On the other hand, different types of SSBs will have different difficult to implement.

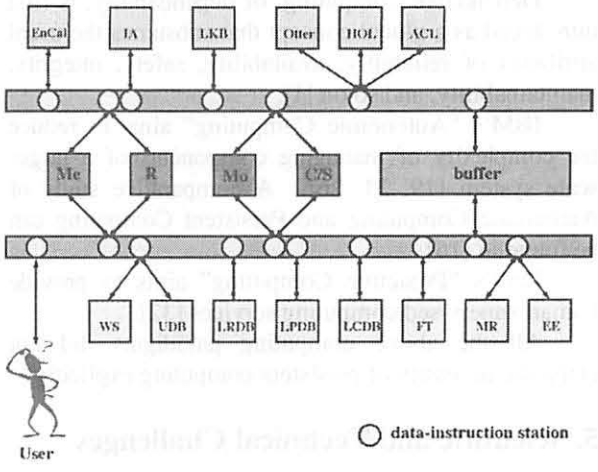

Fig. 1 SSB architecture of HILBERT

As an example, Fig. 1 shows SSB architecture of HILBERT, an autonomous evolutionary information system we are developing for teaching and learning logic [18]. The group of central control components includes a central measurer (Me), a central recorder (R), a central monitor (Mo), and a central controller/scheduler (C/S), all of which are permanent components of the system, and are independent of any application. These central control components are connected by two SSBs such that all data and instructions are sent to or received by components only through the SSBs and there is no direct interaction which does not invoke the buses between any two components. The functional components are measured, recorded, monitored, and controlled by the central control components. All measurement data, instructions issued by the central control components, and communicating data between components flow along the SSB.

As this example shows, in an SSB-based system, the group of central control components can be regarded as the 'heart' and/or 'brain' of the system, the SSBs can be regarded as 'nerves' and/or 'blood vessels' of the system, while the functional components can be regarded as the 'mouth', 'eyes', 'nose', 'hands', and 'feet' of the system.

4. Comparing Persistent Computing with Related Works

Atkinson and his colleagues have proposed the concepts of persistent programming and persistent programming languages in order to store data persistently in database systems [2-4].

Ubiquitous Computing, originally proposed by Weiser, aims to provide the method of enhancing computer use by making many computers available throughout the physical environment, but making them effectively invisible to the user [33].

Dependable Computing, or dependability, is first introduced as a global concept that subsumes the usual attributes of reliability, availability, safety, integrity, maintainability, and so on [1].

IBM's "Autonomic Computing" aims to reduce the complexity of managing components of a large-scale system [19, 23, 26]. A comparative study of Autonomic Computing and Persistent Computing can be found in [16].

Intel's "Proactive Computing" aims to provide human-supervised computing services [32].

All the above computing paradigms did not claim the necessity of persistent computing explicitly.

5. Scientific and Technical Challenges

Until now, there is no computing system that has been implemented to satisfy all requirements for persistent computing systems. Probably, even some implementation issues have not been identified. To implement a true persistent computing system useful in practices, we have to solve many scientific and technical challenging problems.

A scientific and theoretical fundamental problem is how to define a persistent computing model formally. Traditionally, the notion of computability in computer science is intrinsically a finite concept such that any computable problem must be able to be computed within finite steps. However, persistent computing itself as well as anticipatory computing intrinsically concerns an infinite sequence of states to be computed. Thus, from the viewpoint of computation, we have some fundamental questions as follows: What is "computable" by a persistent computing system as well as an anticipatory computing system? Is there some intrinsic difference between the notion of "computability" by persistent computing as well as anticipatory computing and the notion of Turing-computability? Is there some intrinsic difference between the notion of "computability" by persistent computing and that by anticipatory computing? Is any Turing-incomputable problem "computable" by a persistent computing system as well as an anticipatory computing system?

Another scientific and theoretical fundamental problem is how to define the notion of function and the notion of reaction of a computing system formally such that their difference can be used for distinguishing functional states, partially functional states, and disfunctional states from reactive states, partially reactive states, and dead states.

The concept of autonomous and continuous evolution of a persistent computing system also should be clarified philosophically and theoretically.

The biggest technical challenge offered by persistent computing is how to protect, maintain, upgrade, and reconfigure control components of a persistent computing system. Because the control components are the pivot of a persistent computing system, trouble of any control component may lead to a dead state of the whole system.

Any of reliability and security policies, requirements, functions, and facilities of a persistent computing system must be able to be updated, exchanged, added, or deleted while running of the whole system without stopping service. How to satisfy this requirement is a completely open problem.

In order to implement a persistent computing system, the methodology and technology for self-measuring, self-monitoring, and self-controlling are indispensable.

To implement a true persistent computing system, at first we need to have some technical ways previously to test and debug it. A persistent computing system has to be maintained, upgraded, and reconfigured during its continuous and persistent running. This raises a new technical challenge: how to test and debug a persistent computing system running continuously without stopping? Almost all the existing testing and debugging technologies take programs of a system rather than the running system itself as the objects and/or targets. A fundamental assumption underlying the existing testing and debugging technologies is that any program can be executed repeatedly with various input data only for testing and debugging without regard to stopping the task that program has to perform. However, for persistent computing systems this fundamental assumption does not hold no longer. Therefore, we have to find a new way to test and debug a system running continuously and persistently [14].

References

[1] R. J. Abbott, "Resourceful Systems for Fault Tolerance, Reliability, and Safety," ACM Computing Surveys, Vol. 22, No. 1, pp. 35-68, 1990.

[2] M. P. Atkinson, P. J. Bailey, K. Chisholm, W. P. Cockshott, R. Morrison, "An Approach to Persistent Programming," Computer Journal, Vol. 26, No. 4, pp. 360-365, 1983.

[3] M. P. Atkinson and O. P Bunema, "Types and Persistence in Database Programming Languages," ACM Computing Surveys, Vol. 19, No. 2, pp. 105-170, 1987.

[4] M. P. Atkinson and R. Welland, "Fully Integrated Data Environments: Persistent Programming Languages, Object Stores, and Programming Environments," Springer-Verlag, 1999.

[5] A. Avi zienis, J-C. Laprie, B. Randell, and C. Landweh, "Basic Concepts and Taxonomy of Dependable and Secure Computing," IEEE-CS Transactions on Dependable and Secure Computing, Vol. 1, No. 1, pp. 11-33, 2004.

[6] M. V. Butz, O. Sigaud, and P. Gerard, "Anticipatory Behavior: Exploiting Knowledge About the Future to Improve Current Behavior," in M. V. Butz, O. Sigaud, and P. Gerard (Eds.), "Anticipatory Behavior in Adaptive Learning Systems: Foundations, Theories, and Systems," Lecture Notes in Artificial Intelligence, Vol. 2684, pp. 1-10, Springer-Verlag, 2003.

[7] J. Cheng, "Entailment Calculus as the Logical Basis of Automated Theorem Finding in Scientific Discovery," in "Systematic Methods of Scientific Discovery - Papers from the 1995 Spring Symposium," AAAI Technical Report SS-95-03, pp. 105-110, 1995.

[8] J. Cheng, "EnCal: An Automated Forward Deduction System for General-Purpose Entailment Calculus," in N. Terashima and E. Altman (Eds.), "Advanced IT Tools, IFIP World Conference on IT Tools, IFIP96 - 14th World Computer Congress," pp. 507-514, Chapman & Hall, September 1996.

[9] J. Cheng, "The Self-Measurement Principle: A Design Principle for Large-scale, Long-lived, and Highly Reliable Concurrent Systems," Proc. 1998 IEEE-SMC Annual International Conference on Systems, Man, and Cybernetics, Vol. 4, pp. 4010-4015, 1998.

[10] J. Cheng, "Wholeness, Uncertainty, and Self-Measurement: Three Fundamental Principles in Concurrent Systems Engineering," Proc. 13th International Conference on Systems Engineering, pp. CS7-CS12, 1999.

[11] J. Cheng, "Autonomous Evolutionary Information Systems," Wuhan University Journal of Natural Sciences, Vol. 6, No. 1-2, Special Issue: Proceedings of the International Software Engineering Symposium 2001, pp. 333-339, Wuhan University Journals Press, 2001.

[12] J. Cheng, "Anticipatory Reasoning-Reacting Systems," Proc. International Conference on Systems, Development and Self-organization, pp. 161-165, 2002.

[13] J. Cheng, "Temporal Relevant Logic as the Logical Basis of Anticipatory Reasoning-Reacting Systems," in D. M. Dubois (Ed.), "Computing Anticipatory Systems: CASYS 2003 - Sixth International Conference, Liege, Belgium, 11-16 August 2003," AIP Conference Proceedings, Vol. 718, pp. 362-375, American Institute of Physics, 2004.

[14] J. Cheng, "Testing and Debugging Persistently Reactive Systems - A New Challenge in Software Engineering," Proc. Japan Symposium on Software Testing 2005, pp. 34-40, 2005.

[15] J. Cheng, "Connecting Components with Soft System Buses: A New Methodology for Design, Development, and Maintenance of Reconfigurable, Ubiquitous, and Persistent Reactive Systems," Proc. 19th IEEE-CS International Conference on Advanced Information Networking and Applications, Vol. 1, pp. 667-672, 2005.

[16] J. Cheng, "Comparing Persistent Computing with Autonomic Computing," Proc. 11th IEEE-CS International Conference on Parallel and Distributed Systems, Vol. II Workshops (1st IEEE-CS International Workshop on Reliability and Autonomic Management in Parallel and Distributed Systems), pp. 428-432, 2005.

[17] J. Cheng, "Autonomous and Continuous Evolution of Information Systems," in R. Khosla, R. J. Howlett, and L. C. Jain (Eds.), "Knowledge-Based Intelligent Information & Engineering Systems, 9th International Conference, KES 2005, Melbourne, Australia, 14-16 September, 2005, Proceedings," Lecture Notes in Artificial Intelligence, Springer-Verlag, September 2005.

[18] J. Cheng, N. Akimoto, Y. Goto, M. Koide, K. Nanashima, and S. Nara, "HILBERT: An Autonomous Evolutionary Information System for Teaching and Learning Logic," Proc. 6th International Conference on Computer Based Learning in Science, Vol. 1, pp. 245-254, 2003.

[19] A. G. Ganek and T. A. Corbi, "The Dawning of the Autonomic Computing Era," IBM Systems Journal, Vol. 42, No. 1, pp. 5-18, 2003.

[20] D. Harel and A. Pnueli, "On the Development of Reactive Systems," in K. R. Apt (Ed.), "Logics and Models of Concurrent Systems," pp. 477-498, Springer-Verlag, 1985.

[21] H. Hecht, "Fault-Tolerant Software for Real-Time Applications," ACM Computing Surveys, Vol. 8, No. 4, pp. 391-407, 1990.

[22] D. S. Hermann, "Software Safety and Reliability: Techniques, Approaches, and Standards of Key Industrial Sectors," IEEE-CS Press, 1999.

[23] P. Horn, "Autonomic Computing: IBM's Perspective on the State of Information Technology," http://www.research.ibm.com/autonomic/index.html, October 15, 2001.

[24] IEEE-CS, IEEE Standard 610, "IEEE Standard Computer Dictionary – A Compilation of IEEE Standard Computer Glossaries," 1990.

[25] IEEE-CS, IEEE Standard 610.12-1990, "IEEE Standard Glossary of Software Engineering Terminology," 1990.

[26] J. Kephart and D. Chess, "The Vision of Autonomic Computing," Computer, Vol. 36, No. 1, pp. 41-50, IEEE-CS, 2003.

[27] M. R. Lyu (ed.), "Handbook of Software Reliability Engineering," McGraw-Hill, 1996.

[28] J. J. Marciniak (ed.), "Encyclopedia of Software Engineering," John Wiley & Sons, New York / Chichester / Brisbane / Toronto / Singapore, 1994.

[29] A. Pnueli, "Specification and Development of Reactive Systems," in H.-J. Kugler (Ed.), "Information Processing 86," pp. 845-858, IFIP, North-Holland, 1986.

[30] P. Rook (ed.), "Software Reliability Handbook," Elsevier, London / New York, 1990.

[31] R. Rosen, "Anticipatory Systems - Philosophical, Mathematical and Methodological Foundations," Pergamon Press, Oxford, 1985.

[32] D. Tennenhouse, "Proactive Computing," Communications of the ACM, Vol. 43, No. 5, pp. 43-50, 2000.

[33] M. Weiser, "Some Computer Science Problems in Ubiquitous Computing," Communications of the ACM, Vol. 36, No. 7, 1993.

[34] L. Wos, "Automated Reasoning: 33 Basic Research Problems," Prentice-Hall, 1988.

[35] L. Wos, "The Problem of Automated Theorem Finding," Journal of Automated Reasoning, Vol. 10, No. 1, pp. 137-138, 1993.

[36] A. Y. H. Zomaya (ed.), "Parallel & Distributed Computing Handbook," McGraw-Hill, 1996.