APPLICATION-ORIENTED OBJECT ARCHITECTURE: CONCEPTS AND APPROACH

Ramesh K. Karne, Rajavardhan Gattu, Xubo Zhang, and Ramesh Dandu
Department of Computer and Information Sciences, Towson University, Towson, MD 21252

ABSTRACT

The evolutionary computer systems and applications growth has been astronomical and spreads across many dimensions. We propose a radical architecture that is in its conceptual phase, which is based on applications and object-oriented paradigm, instead of computer and system environments. In its initial phases, the application-oriented object architecture has a potential of running on a bare machine without any need for a conventional operating system. This approach bundles the operating system facilities needed with its application program and expects the hardware to communicate with software in an intelligent manner. This paper presents the architecture, identifies its issues related to design and implementation and describes its current state. The proposed architecture is currently demonstrated using existing computer environments and small example applications. Further research is in progress to demonstrate complex examples that can be run on the application-oriented object architecture.

KEY WORDS

AOA, application-oriented, bare machine, computer environments.

1. Introduction

The evolution of information technology can be viewed as having at least eight dimensions: **1.** Vendors (each vendor has their own implementation and possibly different hardware organization) **2.** Hardware releases (frequent product releases due to technology improvements) **3.** Software releases (frequent product releases due to new functionality or improvements) **4.** Operating systems (many types and versions), **5.** Programming languages and tools (proliferation of languages, paradigms, and environments) **6.** Application environments (fast growth in desktop, Internet, multimedia, database, networking, security, etc) **7.** Applications (exponential growth in real-world computer applications) **8.** Secure systems (secure protocols and architectures). As technology evolves in these multiple dimensions, it spawns increasingly complex computer systems and environments.

It can be inferred that current computer systems and environments cause the following problems:

(a) incompatibility among products and tools
(b) obsolescence due to hardware, software, tools and applications
(c) personnel re-training as a result of new environments, programming languages, and tools
(d) wastage of hardware, software, applications, and skills (waste of money)
(e) proliferation of products, languages, interfaces, architectures, platforms, and tools
(f) dumping of obsolete computer hardware and software products (environmental effects, dump sites)
(g) lack of portability, extensibility, maintainability and inter-operability
(h) many layers between applications and hardware resulting in numerous transformations.

It is not proven, but apparent that the above problems may have stem from the evolutionary path chosen to build computer systems over the past few decades. Our motivation emerged from the above problems and frustration and resulted in a simple computer architecture that may possibly reduce above proliferations. It may seem that it is going back to a stone age by revisiting the evolution, however, it may be the only approach available to reduce this proliferation and obsolescence of computer environments.

The first and foremost software system that comes close to hardware is an operating system. An operating system provides services to computer applications and acts as a layer to shield user applications from hardware intricacies. However, an application program to run in a given hardware has all the intricacies of hardware and its interfaces embedded into its program through compilers, linkers, loaders, operating system libraries, device drivers and so on. An application program does not see this complexity but the system does to run in its current environment.

We propose a solution that focus on repackaging the application programs with operating system and environment related programs (bundle them together). That means, there is only one program unit that has the sole knowledge of its creation, execution and retention. The program can also runs on a bare machine so that there is no other program loaded a priori before an application program is run.

2. Related Research

Over the years, related research has explored different paradigms and methodologies to make computing more effective and efficient for computations. Such approaches spread across many fields in computer science, and address some of the problems in current computer systems. In the early 80's, Myers [1] clearly identified the existing semantic gap between computer applications and computer implementations, concluding that it is necessary to reduce this gap to achieve maximum performance and stability in computer systems. The object-oriented paradigm offers many benefits and provides features to extend the existing objects to future objects; object-oriented technology is mature and became the de-facto standard in many areas. Object-oriented programming languages such as Java provide platform independent software and offer portability and reusability. Open-systems and architectures such as Open Software Foundation (OSF), Common Object Request Broker Architecture (CORBA), Enterprise Java Beans (EJB), and Enterprise Frameworks [2] help to develop interoperable systems. The building block approach in software systems [3] provides easier techniques to build and extend software. Dedicated architectures [4] will help in speeding up some specialized applications. However, research has shown that it is very difficult to develop homogeneous software [5] to build interoperable systems, as there is an architectural mismatch in software systems. Exokernel research work at MIT [6] focuses on increasing the performance of applications using Exokernel instead of commercial operating systems (treating operating system like an application). Bare PC research at Utah [7] may eventually help users to write application programs with the interfaces available in OS Kit, which enables PCs to run without major operating systems. The object-engine [8], and OO techniques in hardware design indicate that there is a need for integrated hardware approaches to address current computer system design problems. Virtual computer systems and global software [9,10] provide an alternate approach to cope with the myriad of platforms and millions of computers connected to the Internet. It should be noted that most of these research efforts mesh with the evolution of technology [11] that began decades ago.

The above solutions address design issues such as portability, extensibility, reusability, interoperability, and modularity, yet fail to pay attention to obsolescence and the proliferation of computer systems and applications resulting from the "eight-dimensional cube" (vendors, hardware, software, operating systems, programming languages, application environments, applications, and security). Consequently, despite many technological advances, we still replace our desktops and operating systems every two years, and do not recycle most of our computer hardware and software [12]. The object-oriented computer architecture introduced by the PI in [13], and the global computer architecture described in [14], led to the Application-oriented Object Architecture (AOA) proposed here.

3. Architecture

The proposed *Application-oriented Object Architecture* (AOA) solution is simply a two-layer architecture consisting of an application layer and a hardware layer. The application layer is implemented using application objects (AOs). The hardware layer is implemented using application units (AUs). AOs and AUs communicate through standard interfaces shown in Figure 1. For example, a display AU has its own interface with its AO. A CPU AU has another interface with its own AO. For example, a given application object can request a CPU directly for its time, starting of execution, stopping of execution, availability of CPU, and so on. Similarly, the application units will also keep track of their own resources and interact with AOs. A set of application objects forms a computer application. A set of application units forms a computer hardware system. Some of the features of the architecture are briefly described in this section. Due to space limitations, detailed descriptions and implementation issues are avoided. The major characteristic of this architecture is that it does not require any operating system to be resident in the hardware layer to run AOs.

In this architecture there is a bare machine that can communicate to an application object every time it needs to be executed. The bare machine has to be intelligent to keep track of its own resources and able to communicate with software (AOs). The application object must contain all necessary programs to load, execute and manage its own resources. By bundling all the programs with the application, it will help standardize the application units and avoid obsolescence and proliferation of products. Every AU must follow the standard communication to their AOs. The interfaces to AOs must be built right into AUs to facilitate this communication. The current computer systems do not do this as the AUs have no knowledge of their own resources and they are being managed by the operating system and also there is no well-defined external interface to current AUs as they only interface with the host operating system.

3.1 AO/AU Objects

A set of AOs forms a real-world application. Editing a resume is a real-world application that constitutes a set of AOs that may need to communicate to a set of AUs. A real world application such as editing a resume consists of all knowledge and control needed to execute its application. It is self-controllable and self-executable given required resources or AUs. It is self-controllable as it has all the information related to a given application and it is self-executable as it also sets an execution environment in AUs to run its own application. Thus, an AOA application negotiates with AUs for its resources, load itself, set up its resources, execute itself, monitor errors, process exceptions, and leave the AUs when it is complete. There is no other program in the AUs other than the programs in the AOs.

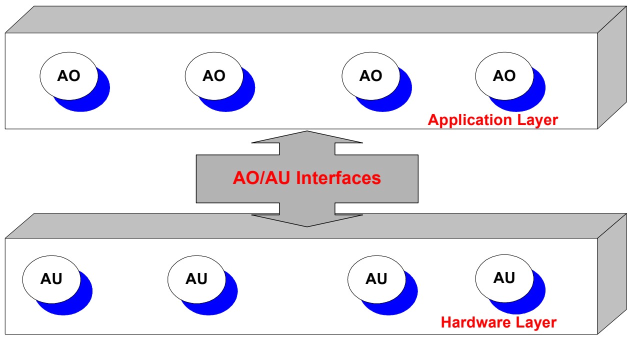

Figure 1. Application-oriented Object Architecture (AOA)

A set of AUs are real-world hardware objects in hardware layer that provides all the required resources for a given application. It is simply a hardware unit consisting of a processor, memory, disk, display, network card and other related elements required for AOs. An AU acts like a server to an AO, in the sense that an AO can make service requests to the AU. Some of the AO's requests are as follows: allocate memory, acquire a CPU for execution (and possibly for a given amount of time), de-allocate memory, put the processor in a wait state, start loading a program, stop the processor, check authentication, validate its signature, and request processor or memory availability. As there is no operating system in an AU, the AU unit and its elements should keep track of their own resources in an intelligent manner. For example, a CPU element or an AU will keep track of its AO currently in execution and the AO's execution time, its interface to the AOs, the amount of time allocated for each AO, whether it can serve a given AO request or not. *If the AUs manage their own hardware resources intelligently, and the AOs have all the necessary software, then these units could function in harmony eliminating the need for an external operating system.*

3.2 AO/AU Interfaces

There is a need for standard interface between AUs and AOs. For example, a keyboard must have a standard interface (similar to BIOS calls) to a keyboard AO. Similary, a CPU should have a standard interface to a CPU AO and so on. Once these standard interfaces are defined, they must only extend to cover new functionality, but not to abandon the old interface. That means, the hardware and as well as software must follow object-oriented methodology in their architecture, design and implementation.

3.3 Example Illustration

The example shown in Figure 2 illustrates a resume application in the AOA system. A resume application has functions such as access control, signature validation, edit, store, and spell-check. In a conventional environment, as shown in Figure 2(a), Microsoft Word, and the underlying OS will enable a user to perform the above functions for the resume application. In the AOA system, as shown in Figure 2(b), an AO consists of a program text object as well as application control objects.

Object-oriented analysis and design could be performed on similar applications such as books, documents, newspapers, magazines and technical journals that may share common objects with the resume object. These similar entities form the basis to build AOs that constitute a specific application domain. In general, we can capture AOs based on applications and hide all the necessary data and control. Also, once all application domains are modeled as application objects, much redundancy will be avoided.

4. Design and Implementation

Today's hardware such as a 386 chip does not provide the interfaces that are required for the AOA. One possible approach is to define the AU component interfaces such as CPU, memory, keyboard, display, network card, mouse, and disk in assembly language or C, letting these interfaces enable communication between AU elements and AOs. However, when such assembly language interface objects are developed to interface with AOs, they should use BIOS calls for hardware functions to make the interfaces stable (assuming that BIOS are part of hardware). The BIOS calls only work in real mode; thus, one must learn how to swap back and forth from real mode to protected mode, if protected mode is used for high memory and 32-bit addressing. We need a clean approach to develop these interfaces that are stable and robust for all application objects.

As shown in Figure 2 for resume object, in general, we need to develop generic interfaces to communicate between AOs and AUs. Once these interfaces are well understood and defined, then it is possible to integrate these interfaces into AUs. For example, a CPU interface can be integrated into the CPU, and a memory interface can be integrated into memory. In this approach, a memory unit knows how much of it is being used and who is using it. It also knows how to communicate with an AO and respond according to its available memory and performance. If the AOA system becomes a commodity, then it is possible to envision custom designs for AUs using a coarse-grained approach, in which multiple elements can be integrated into a single AU, and dynamically re-configured for AO requirements. This means there will be many AUs for specific application objects; such AUs will become a commodity in computer systems. We need further research for a variety of design approaches for AUs, and their pros and cons.

5. Prototype

We have built an AOA prototype using 386 based PC and Visual C++ 6.0 and MASM 6.11 assembler programs. This prototype does not use any operating system related functions except ROM BIOS (we assume ROM BIOS are hardware related and stable). We have modeled about three small applications such as resume, bubble sort, and hello and these applications can run with out any use of DOS or Windows or NT operating system. Some of the high lights are listed below:

- functions for application operating objects for a resume object are identified and implemented
- requirements for AO design and implementation are identified
- AO designs for a resume object and other sample objects are complete
- created a bootable disk whereby the resident operating system can be bypassed
- created protected mode where 32-bit addressing and large physical memory can be accessed
- wrote a loader program which can read programs from a floppy disk and load them into selected locations in memory
- created a user interface for AOA where different application programs (such as a resume object, bubble sort, etc.) can be defined, loaded, and executed in the AU
- implemented all hardware interfaces and operating system type interfaces using ROM BIOS calls
- implemented call gates to provide inter-segment transfers in memory and task gates to create multi-tasking environment
- replaced JMP, CALL, and RET instructions which are linker and loader sensitive with the AOA based interfaces
- found many Web sites and related interest groups (DJGPP, NASM, OSKIT, Exokernel) for free code that works with non-commercial OS environment.

The prototype demonstrates that a C++ application program can run on a bare machine without any operating system resident in the hardware. However, the ROM BIOS need to be resident in the machine.

A bootable disk is created with AOA interfaces where upon booting the machine with a floppy disk will load the AOA interface programs in the memory below 1M range. This AOA interface programs will enable the user to load AOA applications and run them with out any assistance from the standard operating system calls.

In order to do that, the application programs must be written without any *.h files that are related to operating systems. Instead, the programmer must use AOA interface calls.

A real-mode to protected mode and vice versa interface is also developed which will enable the applications to be loaded above 1M and use real-mode facilities such as keyboard, and monitor. An application program can be loaded any where above 1M as specified by the user. When a program is complete, the control is returned back to the user program to terminate.

In protected mode, we have also written AOA interfaces for keyboard, and display using interrupts and BIOS calls.

Currently, all the programs have to be loaded through a floppy disk, however, one can modify the interface to load it from a hard disk.

We are also studying more complex applications such as TCP/IP to run with the AOA system. The prototype is also being used to develop other AO and AU interfaces and serves as a Testbed for AOA applications.

6. Further Research

Initial work in exploration of AOA indicate that there are no major road blocks to pursue this exploration. However, demonstration of AOA using the existing computer environment is a daunting task. We identify the following research areas to make AOA successful in real world applications:

- demonstrate a complex application using AOA
- develop the standard interfaces for AOs/AUs
- study the integration of AO/AU interfaces into AU
- study the issues related to AO communication and sharing of resources
- study the multi-processing issues related to AOA implementation
- study performance and memory needs for AOA.

The above research issues identified requires vast amount of research and funding resources. We believe that when a complex application such as TCP/IP protocol is demonstrated on AOA, then this research will become more fruitful and it will demonstrate a greater potential for laying a foundation for future computer architectures.

7. Conclusions

This paper presented a radical architecture that bundles operating system functions and applications as a single object. The architecture presented is at a conceptual level and only simple applications are demonstrated at this point. The architecture poses many questions than answers, however, we believe that it has tremendous potential to develop a new generation of computer systems and applications for the future.

8. Acknowledgements

This research was supported by National Science Foundation, SGER grant, CCR-0120155. Dr. Frank Anger and Dr. Spencer Rugaber's support and confidence in this project started a first milestone for a possible revolution in computer architectures.

REFERENCES

[1] G.J.Myers, Advanced Computer Architecture (John Wiley & Sons, 1982, 17).

[2]G.Larsen, Component-based Enterprise Frameworks, Communications of the ACM, October 2000, Vol. 43, No. 10, 2530.

[3] F.J.Van Der Linden, and J.K.Muller, Creating Architectures with Building Blocks, IEEE Software, November 1995, 51-60.

[4] G.Borriello, and R.Want, Embedded Computation Meets the World Wide Web, Communications of the ACM, May 2000, Volume 43, Number 5, 59-66.

[5] D.Garlan, R.Allen, and J.Ockerbloom, Architectural Mismatch: Why Reuse Is So Hard, IEEE Software, November 1995, 17-26.

[6] D.R.Engler, The Exokernel Operating System Architecture, Ph.D. Thesis, MIT, October 1998.

[7] B.Ford, G.Back, G.Benson, J.Lepreau, A.Lin, and O.Shivers, The Flux OSKit: A Substrate for OS and Language Research. In Proc. of the 16nth ACM Symp. on Operating Systems Principles, St. Malo, France, Oct. 1997, 38-41.

[8] F.Cummins, R.Cunis, and G.Harris, The Object Engine: Foundation for Next Generation Architectures," OOPSLA '95, 123-127.

[9] J.Z.Gao, C.Chen, and D.Leung, Engineering on the Internet for Global Software Production, Computer, May 1999, 38-47.

[10] A.S.Grimshaw, Wm.A.Wulf, and the Legion team., The Legion Vision of a Worldwide Virtual Computer, Communications of the ACM, January 1997, Vol. 40, No. 1, 39-45.

[11] J.Hennessy, The future of systems research, Computer, August 1999, 27-33.

[12] J.Whitley, A Study of Computer Obsolescence and Its Impact, M.S Thesis, Department of Computer and Information Sciences, Towson University, Towson, MD 21252, December 2001.

[13] R.K.Karne, Object-oriented Computer Architectures for New Generation of Applications, Computer Architecture News, December 1995, Vol. 23, No. 5, 8-19.

[14] R.K.Karne, and J.Bradley, A Global Computer Architecture: A Revolutionary Approach For Global Computing of the Future, AoM/AΦM 14th Annual International Conference, Conference Proceedings, Toronto, Canada, August 1996, 213-225.

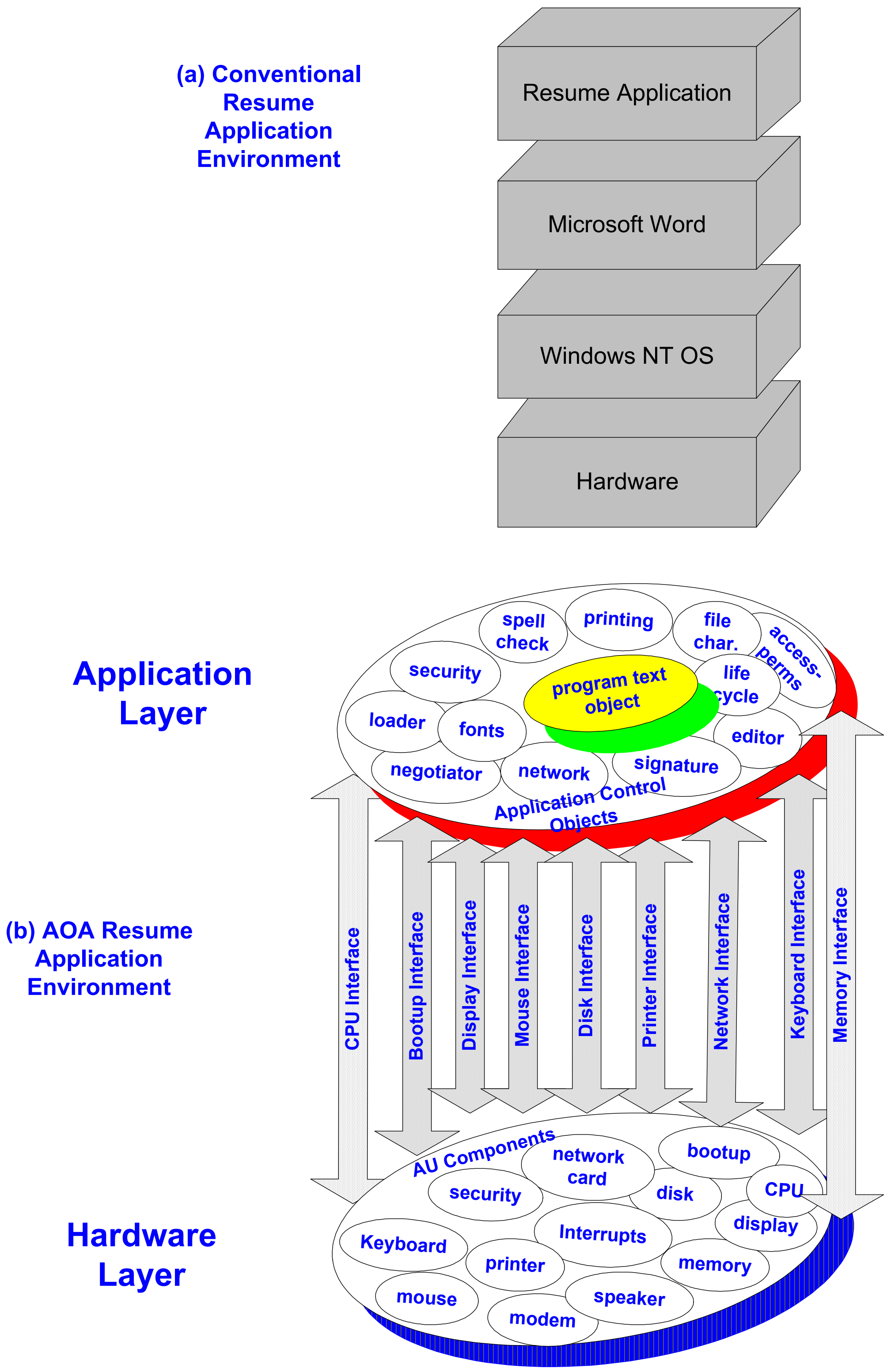

Figure 2. Resume Object