DOSC: Dispersed Operating System Computing

Ramesh K. Karne
Towson University
rkarne@towson.edu

Karthick V. Jaganathan
Towson University
kvenka1@towson.edu

Tufail Ahmed
Towson University
tahmed3@towson.edu

Nelson Rosa Jr
Dartmouth College
nr@dartmouth.edu

ABSTRACT

Over the past decade the sheer size and complexity of traditional operating systems have prompted a wave of new approaches to help alleviate the services provided by these operating systems. The emergence of micro-kernels and a plethora of non-traditional operating system models, both geared toward reducing the role of the OS, attest to the promise of practical alternatives.

The problem with these methods is that the three-tiered system of software, operating system, and hardware is still preserved. Even though the operating system might find some reprieve by having to handle less work there is a nascent notion being triggered by these alternative approaches that the operating system as an abstract entity is no longer a necessity. We propose a radical method of computing where we take this notion to the extreme and push the operating system into the software and hardware levels. By doing so, we create a decentralized operating system environment known as Dispersed Operating System Computing (DOSC). We outline how the Dispersed Operating System paradigm works, its benefits, and immediate practical applications in today's world.

Categories and Subject Descriptors
D.4 [**Operating Systems**]: Miscellaneous.

General Terms
Design

Keywords
Operating Systems, Application Object, Dispersed Operating System Computing (DOSC), and Object-oriented.

1. INTRODUCTION

The traditional operating system has come under attack over the last decade as an entity that is no longer providing services that make computer systems convenient and efficient. With over thirty million lines of code [23] and an ever increasing amount of complexity [22], today's popular operating systems continue to only harm the applications they run. Each piece of functionality not used only burdens an application with unwarranted performance penalties, short-lived code, and adherence to a specific operating system platform. Contrast this to the hardware where relative to the operating system, is fast, standardized, stable, and very cheap, the operating system as an abstract layer comes into question. In fact, many alternative approaches to the traditional operating system have been successfully implemented in both hardware and software with the increasing trend that the operating system does not have to do everything. The problem with these approaches is that they do not go far enough. They still preserve the concept of an operating system as a necessary agglomeration sitting in-between hardware and software. Two papers of note, [1] and [5], list plenty of reasons as to why the operating system should cease to exist as a centralized unit.

Furthermore, embedded systems [2] and pervasive computing [14] are becoming more commonplace and the traditional operating system will not be able to adequately cover the customized set of services needed for these varying computing systems. We posit that in the end only a direct relationship between hardware and software will yield the best results and that a new paradigm has to be introduced to fully capture such a novel environment. The Dispersed Operating System paradigm is what will allow application-centric computing that is object-oriented to come to fruition. The DOSC model takes full advantage of the hardware through methods of encapsulation. Throughout the remainder of the paper we further expand on the DOSC paradigm, propose a viable infrastructure to support the DOSC environment in hardware and software, current implementations of DOSC computing, and where future research will take us.

2. RELATED WORK

The closest related works to our approach are the OSKIT [8, 19] and the Exokernel [4, 5]. Both of these approaches break down the operating system into its necessary components and dilute the centralized importance of a traditional OS. Despite showing the performance benefits of a trimmed OS there still lacks a proper investigation into seeing how applications can benefit without an OS present. The Service-Oriented Computing (SOC) [16] architecture is a good conceptual model on sustaining application governed systems. The DOSC paradigm tries to parallel the SOC approach in the DOSC abstraction of hardware and software. The SOC only applies to e-commerce and so has a limited scope in its application. The feasibility of running an object-oriented operating system on a bare machine was shown by The Choices project [20]. This furthered the conviction that a complete object-oriented environment was possible on a bare machine. Sandboxing mechanisms [15] allow applications to configure and deploy services at user level, which suggests that performance gains are possible when functionality is moved from the OS level to the application level. Other related works such as micro-kernels [7], SPIN [3], server operating systems [9], embedded web server library SWILL [13], Plan 9 [17], nano-kernel [21], VM and Virtual PC [6], and similar concepts take advantage of the hardware and software to provide improved OS services. The DOSC paradigm strives to improve upon all of these devices by providing an efficient link between software and hardware, such that their performance can only be improved by the removal of the traditional OS.

These works have provided great motivation to take the operating system one step further into disseminating the OS into hardware and software. This is a necessary step because all of these different architectures do not go far enough and have not adequately explored the frontier of computing by removing the OS as an entity.

3. THE DOSC PARADIGM FOR SOFTWARE

The DOSC paradigm takes the operating system as an entity and moves the OS services into hardware and software. The computer is in essence a bare PC, but this makes an application the cornerstone of the computer system. The software layer is abstracted into an Application Object (AO), which encapsulates the application(s) and the Application Operating Environment (AOE) [11]. By following these tenets of DOSC computing the hope is to design applications for long term stability, performance improvements, and achieve savings of time and money through code reuse and extensibility.

3.1 The Application

Applications are developed using object-oriented programming. An object-oriented model for developing applications encourages modularity and the use of inheritance to extend an applications lifespan. Also, applications become components that are encapsulated together within an AO. This allows the programmer the necessary level of abstraction to include only the necessary components needed to implement the final product. It is important to stress that an Application Object is not restricted to just one application and can encapsulate many applications at once. A complex AO could be a banking application or a desktop application where multiple applications can work in unison. The AOE provides the environment, which the programmer optimizes, to meet the demands of the enclosed applications.

3.2 The Application Operating Environment

The OS services that are moved into software are bundled together into the Application Operating Environment. The AOE is in charge of managing all resources needed by the applications. The AOE may be implemented in any way the programmer sees fit. Hence an AOE is custom-fitted to meet the demands of the applications found in the AO. A fully adjustable AOE allows the programmer full control of the machine. The programmer can set up any environment that is necessary to give the user the best experience possible. The entire design process is open and allows for the utmost creativity in organizing and deploying an application from the ground up. Although this places more responsibility on the programmer, the AOE provides unprecedented freedom to optimize the application by using the hardware to create the proper environment for the applications.

3.3 The Application Object

In the DOSC way of computing the application and the Application Operating Environment are encapsulated within an Application Object [10, 12]. The AO is the entire software package and must be self-contained, self-managed, and self-executed (SSS) across any bare PC. An Application Object is considered self-contained when all the necessary code, both at the application and environment level, has been encapsulated into a single compiled unit. The AO is self-managed when it provides all of the needed services not found in hardware implemented in the AOE. Finally, an AO is self-executable when it can govern its entire execution flow. This includes loading itself into the system, running the applications, error-handling, and termination of applications and of itself. A concrete example of what an AO would look like at the end of a development cycle is shown in Figure 1. The figure depicts an e-mail AO. This e-mail application is capable of sending and receiving messages using the Simple Message Transfer protocol. The actual implementation of the application is in C++. The AOE enclosed in the AO provides for all of the resource management allowing the e-mail AO to avoid operating system related calls.

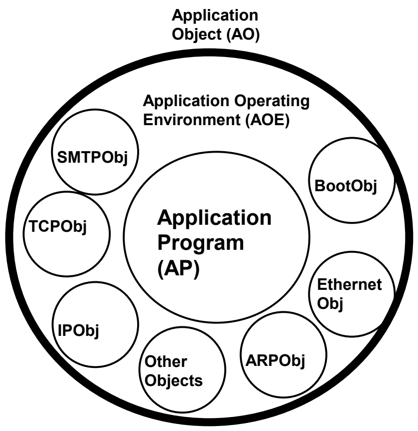

Figure 1. Example AO (Email Client).

4. THE DOSC PARADIGM FOR HARDWARE

The goal of the DOSC paradigm for hardware is to encourage direct high-level calls of the primitives needed to interact with the device. The hardware manufacturer is free to implement the hardware API, but must make the API publicly available and in addition must list all OS services found in the hardware device. Furthermore, every device must have a processor complex. A processor complex is comprised of a processing unit, volatile memory, and a transceiver. A processor complex provides an easy way to integrate all computing systems from mainframes to pervasive computers. These requirements are quite feasible due to the reduced costs of processors, memory, and transceivers.

4.1 Processor Complex

At the heart of the DOSC hardware paradigm is a processor complex. The processor complex is comprised of a processing unit, volatile memory, and a transceiver. The processing unit determines whether the devices qualify as a computing system or an optional I/O device. The computer architecture shown in

Figure 2 displays a computing system. Each computing system is required to have a CPU as the processing unit. For example, a computing system processor complex would be found in desktops, laptops, PDAs, cell phones, and network appliances. The optional I/O peripherals would have a device controller or similar administrative processing unit for operating the device. The controller can be on the hardware unit or as a separate piece of hardware. The controller would allow for the software to communicate with the device through the use of simple function calls.

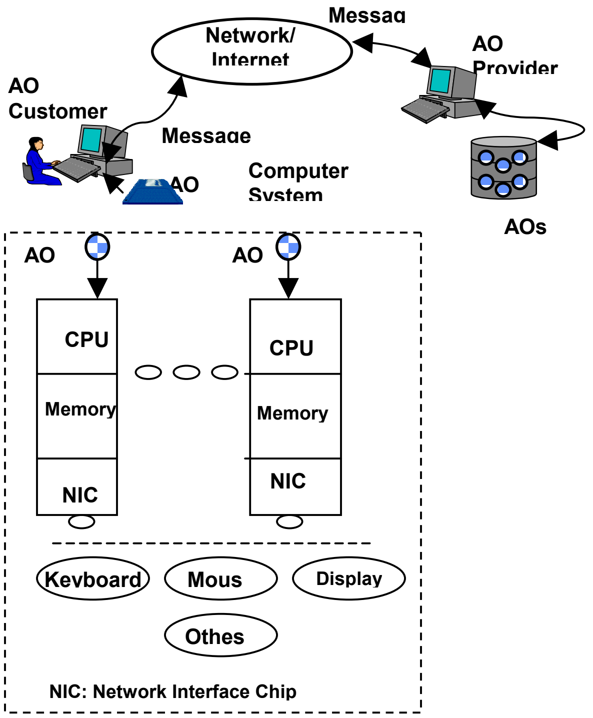

Figure 2. System Architecture Overview.

4.2 Hardware API

The AO to hardware interfaces provide direct communication to hardware through the processing unit present in the hardware device. In effect the device driver is expected to be implemented somewhere inside the processor complex of the device. The API is meant to help promote manufacturers to publish a listing of OS services dispersed into hardware. The Hardware API instructs the software developer as to what the function calls are and what parameters to pass to operate the target device. A sample interfaces are shown in Figure 3.

4.3 AO Control of Hardware

When an Application Object is executing the AO has sole control of the CPU processing complex. This is to avoid the complexities and penalties in context switching such a large amount of state. This might seem as a setback, but the price of a adding an additional CPU processor complex to the computing system is not expensive. The price would conceivably be offset by the purchase of the latest version of a commercial operating system every time a new release became available. On the other hand I/O devices do not need a one to one correspondence with an AO, because arbitration algorithms can determine which AO has access to the device. Of course this is just an implementation detail and if the DOSC paradigm becomes an active area of research several AOs can one day run off of the same CPU processing complex.

5. DESIGN AND IMPLEMENTATION OF SAMPLE APPLICATION OBJECTS

Currently, we cannot expect hardware to provide direct interfaces to AOs. Thus, we have implemented hardware interfaces using the C++ API as shown in Figure 3 and so have resorted to demonstrating the feasibility of the DOSC concept. The hardware platform used for the AOs has been IBM compatible PCs with Intel Pentium processors, a 3COM 3C509B NIC, and at least 128MB of memory. Each AO interface is written in C++ as a method (Figure 3 shows example calls for the sample AO interfaces). As the code is C++ compatible we can take advantage of most standard compilers and linkers to build the Application Object. In order to deploy the AO on a bare PC a basic interface program was written to load the AO. This simple interface provides total control of the system to run AO applications. We eventually plan on removing the dependency of the AO loader and develop the AOs to be fully compatible with the SSS properties outlined for AOs. Initially we programmed a set of simple AOs. These programs were the hello world program, the merge sort and bubble sort algorithms, and an SMTP protocol capable of sending and receiving messages. To further validate the paradigm we have also implemented an e-mail AO and a web server AO as examples of more complex applications in terms of resource management. The API is provided for these AOs to serve as a concrete example on how the DOSC paradigm can simplify the direct communication to the hardware. The API also demonstrates the amount of flexibility granted without a centralized OS.

5.1 An E-mail Application Object

An e-mail application object requires the implementation of the SMTP, TCP, IP, ARP, and Ethernet protocols. All of these protocols had to be re-implemented from scratch because commercially available protocol suites for networking are complex and are not meant to run on bare machines. We have only implemented the generic facilities that are needed for the e-mail client to function correctly. The features implemented in the e-mail client are the typical source address, destination address, subject fields, as well as, attachments and text-only message content. The e-mail client is written as a single threaded model to send messages. However, nothing precludes us to run the client as a multithreaded program.

The e-mail client connects to the Towson University e-mail server to forward the messages. Similarly, the e-mail client receives e-mail messages through the same server using the Post Office Protocol. Figure 4 shows part of the implementation and message flow to send an e-mail. This logic includes: resolving the ARP addresses, making connection to the server, processing packets through Ethernet to the SMTP level, and interfacing with the user. The client does not respond to unnecessary requests coming from the network and simply discard messages that are not relevant to the e-mail AO. The size of the e-mail AO executable is 32,768 bytes with 2,666 lines of C++ code. This count does not include comments. The connection establishment time for the e-mail client to the server is about 3.5 seconds. The connection timecan be dramatically improved if we implemented our own e-mail sever because the campus e-mail server's response is very slow.

AOAIOObject:

```
// get a character from the keyboard
ch = io.AOAgetCharacter(); // INT 0F5h
// print a character at a given location on the screen
io.AOAprintCharacter(ch, 300); // INT 0F7H
// print a hex value at a given location on the screen
io.AOAprintHex(integ, 340);  // INT 0FDH
// print a hex value at a current cursor location on the screen
io.AOAprintHex(integ); // INT 0FDH
// get a string  from the keyboard
int len = io.AOAgetString(buffer); // INT 0F5H
// print a string at a given location on the screen
io.AOAprintString(buffer, len, 400); // INT 0F7H
// print a string at a current cursor location on the screen
io.AOAprintString(buffer, len); // INT 0F7H
//clear screen
io.AOAcleanScreen(); // INT 0F1H
//get cursor position
pos = io.AOAgetCursor(); // INT 0F2H
//set cursor position
io.AOAsetCursor(integ); // INT 0F3H
// read a floppy disk, one sector at a time
io.AOAreadFloppy(buffer, 21); // INT 0FAH
// get a timer value
integ = io.AOAgetTimer(); // ;INT 08H
// exit from program
io.AOAExit();
```

AOAEtherObject:

```
Init(); //  Initialize device
Send(char*, int); // // Send packet
Receive(char*); //  Receive pakcet
Close();  //Close device
getMac(char*); // Return MAC address
```

AOAUDPObject:

```
setTarget(char*, short); // Set target machine
setData(char*, int); // Set data to be sent
Send();      // Send data gram
Receive(char*);          // Receive data gram
Close();      //Close connection
```

Figure 3. Sample AO Interface Calls.

5.2 A Web Server Application Object

In the DOSC paradigm the philosophy is to customize an application's environment so to avoid including unnecessary functionality. The object-oriented approach makes programming in the DOSC environment very efficient because only necessary services need be compiled into the Application Operating Environment. The AOE components are expected to be reusable for use in a variety of AOs. As can be seen from e-mail message flow (Figure 4) and web server message flow (Figure 5) the functionality that is required for the e-mail AO and the web server AO are similar in some areas and different in others. For example, the web server design overlaps with the e-mail design in its implementation of the HTTP, TCP, IP, ARP, and Ethernet networking protocols. Therefore these components can be reused in the web server AO. The web server environment for which these protocols will be used in will be very different from that

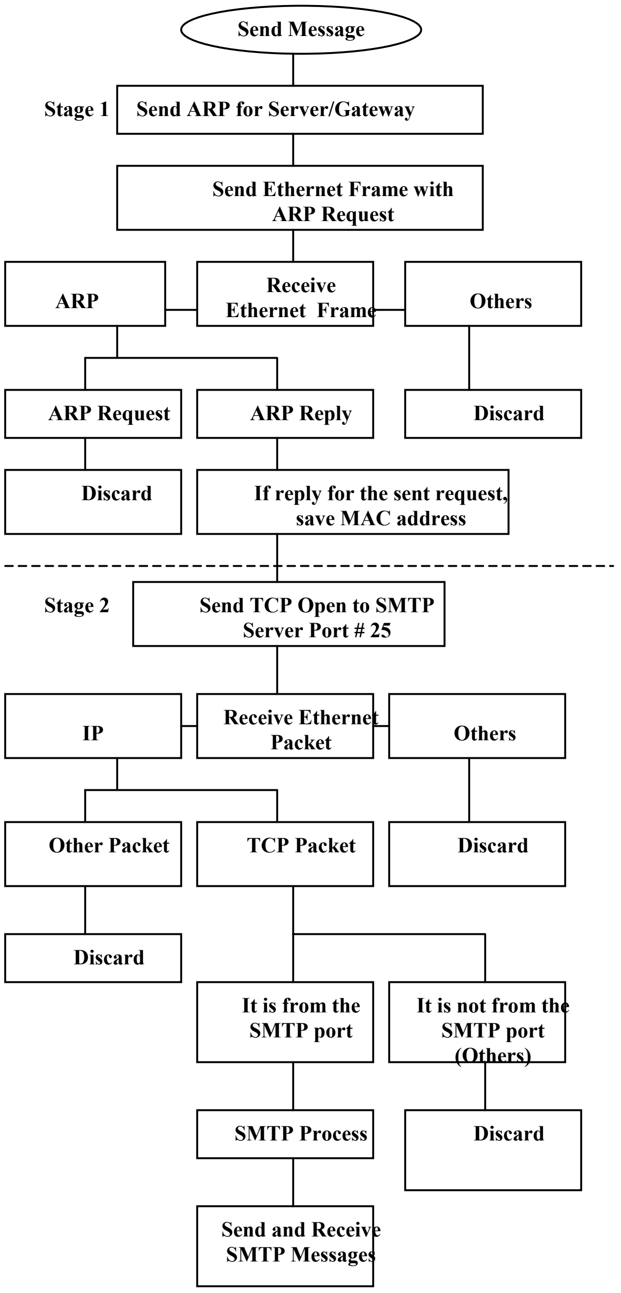
Send Message
Stage 1
Send ARP for Server/Gateway
Send Ethernet Frame with ARP Request
ARP
Receive Ethernet Frame
Others
ARP Request
ARP Reply
Discard
Discard
If reply for the sent request, save MAC address
Stage 2
Send TCP Open to SMTP Server Port # 25
IP
Receive Ethernet Packet
Others
Other Packet
TCP Packet
Discard
Discard
It is from the SMTP port
It is not from the SMTP port (Others)
SMTP Process
Discard
Send and Receive SMTP Messages

Figure 4. Email Message Flow.

found in the e-mail environment. A web server has to constantly listen to the client's requests, resolve ARPs, receive Ethernet frames, and send the frames to the appropriate recipient. In the end the network protocol suite is tailored to fit the web server application.

Running a web server on a bare PC was quite a daunting task as it posed numerous challenges in server architecture. The object-oriented methodology used throughout the exploration helped to achieve this milestone smoothly. The web server is fully operational and can be accessed at http://dosc.towson.edu. The web server runs on a Pentium II, 350 MHZ, with 128MB memory with no hard disk and only supports HTTP requests. The server is hosting one of the author's web site as well as over 673 files with over 40MB of file storage. The size of the executable is 77,824 bytes, while the number of lines of C++ code, not including comments, is 6,587. The DOSC prototype environment and AOs will be made available through the web server in the near future. The next step is to conduct performance measurements and benchmarks of the web server.

6. IMPACT OF THE DOSC APPROACH

The DOSC approach is projected to have the following impacts:

- Common applications such as sorting, e-mail, web server, gateways, routers, web browsers, desktop applications, network monitors, sensor networks, security monitors, and more can be better implemented with the DOSC approach. Since the system is bare these applications will be able to run on older machines. This will help reduce the frequent dumping of computers.
- Many common applications (like those mentioned above) developed using the DOSC approach can easily be executed on embedded systems that also adhere to the DOSC paradigm, for example, cell phones. This will save tremendous software development cost, reduce heterogeneity, and avoid middleware to integrate these systems.
- Users can set up their own servers, as the web server and e-mail AOs are fairly small and host their own web sites, e-mail clients, and e-mail servers. This will reduce the tremendous load and traffic on current web and e-mail servers.
- The DOSC approach will encourage hardware vendors to provide standard interfaces. Thus resulting in common interfaces for a variety of pervasive devices. This would make pervasive computing much simpler.
- The DOSC approach can improve the performance of computation intensive applications, as there is no OS overhead in the system. Preliminary results with bubble sort and merge sort on 500,000 elements have yielded that DOSC runs the algorithms 16% faster compared to running on a Red Hat Linux WS4.
- Simple designs that consistently use a small number of general mechanisms [18] such as our e-mail client and web server should be more secure than conventional systems. In particular, the web server AO does not require any firewall to filter ports because it only accepts messages on the standard HTTP port. Intruders will not be able to understand and predict the implementation details and thus making the clients and servers less vulnerable to OS exploits and related attacks.
- As there is no central OS running, AOs do not leave any trace of their execution. This approach can be used to run military or secure applications where two AOs can communicate through their own private protocol without publicizing them to gain more security.

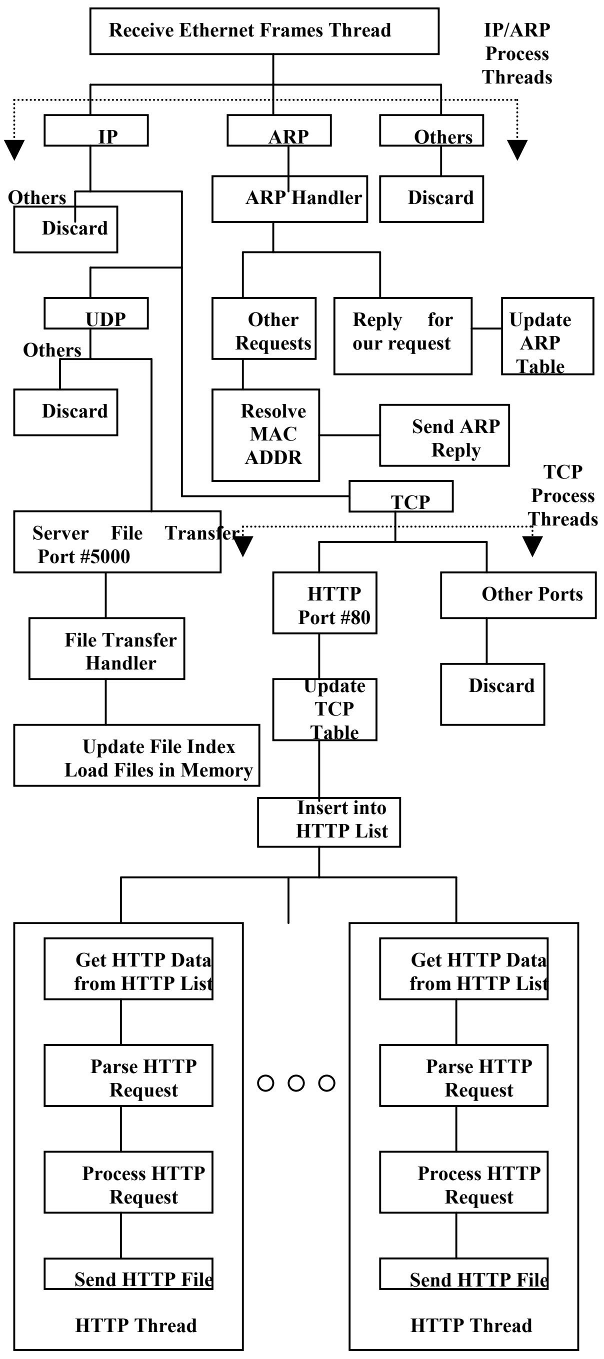

Figure 5. Web Server Message Flow.

The DOSC approach, although still under development, suffers from the following drawbacks:

- There is redundant code in each AO; the common code among AOs is not utilized (if the communalities utilized then it leads into a traditional OS).
- AO programmers have to deal with more complexity in the DOSC approach. On the other hand, it is better for the AO programmer to deal with the nitty-gritty of a system rather than delegating it to an OS middleware, where it may or may not be trusted.
- Developing standards for AOs in software and hardware may not be practical in today's market driven world. However, for certain applications, this approach will be ideal regardless of trends in the market.

7. FUTURE RESEARCH

Currently, we have a complete prototype environment for developing and testing the DOSC applications. In the near future we plan to conduct performance benchmarks and measurements of our web server and compare them with other commercial web servers such as IIS, Apache and Zeus. The followinglist identifies other future research directions:

- Identify design issues related to using the same AO developed for the web server in a cell phone or a network appliance.
- Study the security issues related to the web server AO.
- Develop a VoIP Phone application based on the DOSC paradigm.
- Investigate how AOs can be used as mobile agents to address a variety of middleware problems.
- Investigate how the DOSC paradigm will simplify the IT industry and change the players in the IT arena. This work requires at least the investigation of the following issues:
    - How IT industry will be organized if the DOSC paradigm becomes successful?
    - How should the IT players interact with each other?
    - Who owns the rights of the AOs?
    - How are AOs distributed, controlled, and managed for users?
    - How AOs work with existing applications?
    - How a non-AO environment works within an AO environment?

8. CONCLUSIONS

This paper presented the conceptual foundation for Dispersed Operating System Computing. The DOSC approach is meant to allow software and hardware to directly talk with one another without the middleware operating system. The software is stored inside a self-contained, self-managed, and self-executed Application Objects capable of running on any bare machine. The hardware in the DOSC paradigm contains a processor complex, which allows for easy integration of all devices through a processing unit, memory, and transceiver. The DOSC paradigm can be implemented as proven by the e-mail client AO and web server AO. The DOSC approach could usher in a new wave of computing through its exploration of developing hardware and software on bare systems.

9. ACKNOWLEDGMENTS

We sincerely thank NSF and in particular the late Dr. Frank Anger who initially supported this work by funding SGER grant CCR-0120155. Without his support, our work would have had no opportunity for study and could not have reached this point. We also thank Dr. Joyce C. Little for her valuable advise and input to the paper in its contents and style. Finally, we would like to acknowledge our Summer REU 2004 students Payal Aggarwal, Sean O. Bender, and Mark Arthur Dencler for helping with the initial proof of concepts of the DOSC paradigm.

10. REFERENCES

[1] Anderson, E., Thomas, The Case for Application-Specific Operating Systems, Third Workshop on Workstation Operating Systems, pp. 92-94, April 1992.

[2] Borriello. G., and Want, R. Embedded Computation Meets the World Wide Web, CACM, Vol. 43, No. 5, May 2000.

[3] Bershad, B.N., Savage, S., et al. Extensibility, safety, and performance in the SPIN operating system. In 15th ACM SOSP, pages 267-284, December 1995.

[4] Engler, D. R., The Exokernel Operating System Architecuture, Ph.D. thesis, MIT, October 1998.

[5] Engler, D. R., M.F. Kaashoek, "Exterminate all operating system abstractions," FifthWorkshop on Hot Topics in Operating Systems, p. 78, 1995.

[6] Exploiting Virtual PC, http://www.expresscomputeronline.com/20040607/techspace01.shtml.

[7] Ford, B., Hibler, M., Lepreau, J., McGrath, R., and Tullman, P., "Interface and execution models in the Fluke Kernel," Proceedings of the Third Symposium on Operating Systems Design and Implementation," 1999, pp. 101-115.

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

[9] Kaashoek, M.F., Engler, D.R., Ganger, G.R., and Wallach, D.A. Server Operating Systems, In the Proceedings of the 7th ACM SIGOPS European Workshop: Systems support for worldwide applications, Connemara, Ireland, September 1996, pages 141-148.

[10] Karne, R.K., Gattu, R., Dandu, R., and Zhang, Z., "Application-oriented Object Architecture: Concepts and Approach," 26th Annual International Computer Software and Applications Conference, IASTED International Conference, Tsukuba, Japan, NPDPA 2002, October 2002.

[11] Karne, R.K. Application-oriented Object Architecture: A Revolutionary Approach, 6th International Conference, HPC Asia 2002, December 2002

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

[13] Lampoudi, S., and Beazley. D.M. SWILL: A Simple Embedded web Server Library, Proceedings of the FREENIX Track: 2002 USENIX Annual Technical Conference, Monterey, California, June 10-15, 2002.

[14] Oxygen Project, MIT, http://oxygen.lcs.mit.edu.

[15] Qi, X., Parmer, G., West, R. An Efficient Endhost Architecture for Cluster Communication Services, to appear in Cluster 04.

[16] Papazoglou, M. P., and Georgakopoulo, D., “Service-Oriented Computing”, Communications of the ACM, October 2003, Vol.46, No.10, p.25-28.

[17] Pike, R., Pressotto, D., Thompson, K., and Trickey, H. “Plan 9 from Bell Labs”, UKUUG Proc. Of the Summer 1990 Conf., London, England, 1990.

[18] Saltzer, J. H, and Schroeder, D. The Protection of Information in Computer System. Proceedings of the IEEE 63(9): 1278-1308, September 1975.

[19] “The OS Kit Project,” http://www.cs.utah.edu/flux/oskit.

[20] The Choices: Object-Oriented Operating System, http://choices.uiuc.edu/choices/choices.html.

[21] Tan, See-Mong., Raila, D.K., and Campbell, R.H. An Object-oriented nano-kernel for operating system hardware support, Objectorientation in Operating Systems, 1995, Fourth International Workshop, p220-223.

[22] http://www.wired.com/news/linux/0,1411,66022,00.html. Linux: Fewer Bugs Than Rivals, Dec 14, 2004.

[23] http://www.dwheeler.com/sloc. Estimating GNU/Linux Size, July 2002.