Aspect Oriented Operating Systems

Tzilla Elrad, Paniti Netinant
{elrad, netipan}@iit.edu

We would like to argue that building aspect oriented operating systems is a potential technology needs and promising solutions that can make a significant impact on the way we deign and produce software in the coming future. At the same time, the open, unresolved issues on the basic architecture and design of aspect oriented operating systems puts such adventure beyond the scope of today's short term profit driven research and development programs.

Separation of concerns is one of the crucial principles in software design. It is in the heart of software usability and re-usability. Success of object-oriented programming is largely due to the structured way in which it enables to separate the concerns, decompose the deferent concerns and at the same time protect their integrity.

Success of object-oriented programming languages lead to a whole culture of object oriented software development and finally, research on objects oriented operating systems and realization of such systems.

AOP-Aspect oriented programming has already closed the gap and has matured into AOSD-Aspect-oriented software development. Here we would like to support our proposal why we believe that efforts should be put into exploring the potentials of designing aspect oriented operating systems.

- Applications that can mostly benefit from aspect orientation are those that exhibit a high degree of crosscutting concerns. Operating systems are notorious of their problematic structure due to the high degree of crosscutting concerns.
- Distributed systems –most of the challenges listed on this call for white paper proposal such as partial failure, causal ordering, dynamic service partition, QoS control, multi-level distributed resource management are all aspectual properties - hence a distributed system software design that localizes and encapsulates these aspects has a potential to ease the challenge in this direction.
- Future operating systems would need to exhibit a higher degree of adaptability. Many features that need to be adaptable are aspectual in nature; with current state of the arts, their implementation crosscuts other features and this is a curial barrier for adaptability.
- Reflection – Growing need in intelligent operation systems that wisely manage and adapt to new policies can be practically realized only if implementation of these policies is localized and support conditions and intelligence under which such policies can be evoked.
- Integration of real-world physical constrains – Operating systems often hide the detailed implementation of their services. This is a double edge sword – a specific real-world application may need a specific service implementation tailored to support it own constrains. It is beneficial to open those system implementation details in a systematic and protective way.

Aspect-oriented operating systems should be designed as such. This is more then adding one or two aspectual properties to an existing system. A great deal of research has yet to be invested into the nature of such design.

We attach our paper “A Layered Approach to Building Open Aspect-Oriented Systems “ published at the October issue of the CACM.

A Layered Approach to Building Open Aspect-Oriented Systems

Paniti Netinant[1, 2], Tzilla Elrad[2], and Mohamed Fayad[3]
{netipan, elrad }@iit.edu Fayad@cse.unl.edu

System software is characterized by two important constraints: As the underlying foundation for applications, it is expected to present a relatively unchanging interface while nevertheless adapting to the tempest of technological changes below it. System software must juggle the implementation of a large variety of often interfering and crosscutting concerns such as synchronization, scheduling, fault tolerance and logging. A typical operating system will need sections of code responding to each of these requirements. That code is not usually segregate into neat modules, but instead interacts and tangles with the underlying functionality. System developers want to be able to evolve this code easily as new services are demanded, new devices are invented, and new protocols are implemented. An open system is characterized by its ability to modify the semantics of existing policies and by its ability to add new policies.

Systems are often implemented using the layering approach [5]; a familiar example of such layering is network protocol stacks. In this article we describe the Layered Aspect Moderator Framework; an extension of our prior work on framework-based Aspect-Oriented Programming [4] to the development of open systems software.

Open Layered Aspect-Oriented Systems: Architecture

Systems can be modularized in many ways. One of the most popular is the layered approach. The system is decomposed into a number of layers (levels). Each layer relies on the functionality and interface of the layer just below it. In operating systems the bottom layer interfaces with the hardware; the top layer provides the interface for applications. Layers provide abstraction: a layer does not need to know how the operations of the layer below it are implemented. This has a great advantage in information hiding. However, information hiding is a double edged-sword for aspectual properties such as security, performance, and scheduling. Some implementation parts of the underlying, hidden implementation might eventually be critical to support a specific aspect that emerges later during a lifetime of a system. An application might need to be aware of these implementation details. Moreover it might need to overwrite some of these properties. To overcome the hidden information duality we propose the *Open Layered Aspect-Oriented System Framework.* Our approach aims at supporting two dimensions of open system implementation. The first dimension manages intra-layer compositions. Each layer separates services and their aspectual properties. Each aspectual property, such as scheduling, may have more than one possible policy to realize it. Moreover, each aspectual property is extensible with evolving new policies. A moderator (its design will be explained later) negotiates weaving intra-layer services and aspects at runtime. The horizontal branch of our framework is responsible for this dimension of intra-layer coordination. The second dimension is an open layers interface supporting inter-layers compositions. Components of a higher layer can call services on the neighboring lower layer with a particular choice of aspects provided. For example, a file service can be called with either a FIFO scheduling policy or priority scheduling policy assuming these policies are provided in the lower layer. The inter-layer negotiation is achieved though the vertical branch of our framework. Both the intra-layer and the inter-layer branches provide dynamic adaptability to system aspectual properties.

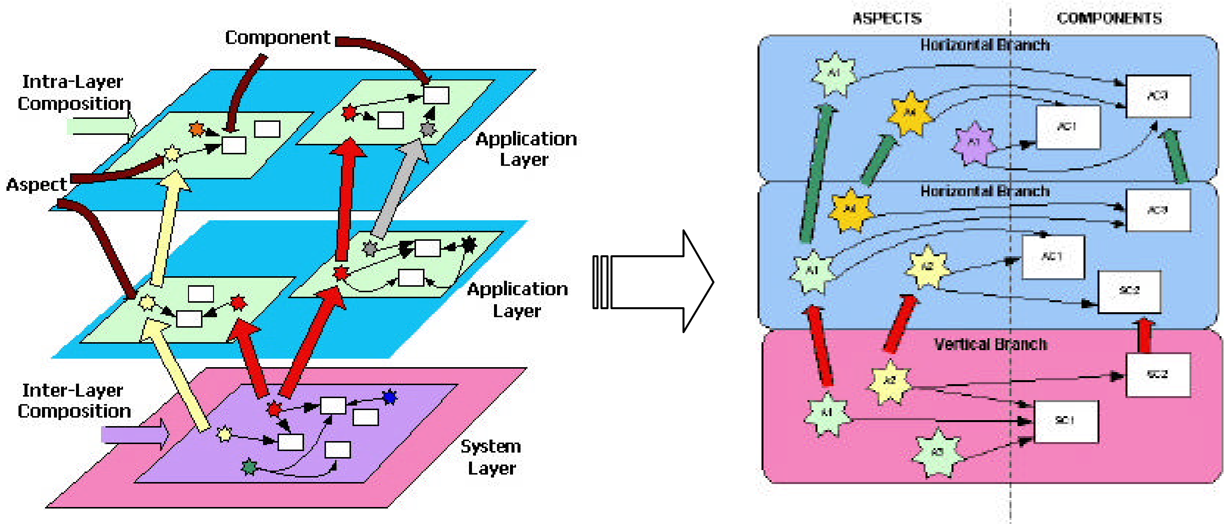

Figure 1. The Architecture of The Layered-Aspect-Oriented System (OL-AOS)

A layer consists of components and aspectual properties controlled by the Asprct Moderator using horizontal branches of the framework as illustrated in Figure 1 on the left. A system consists of layers and inter-layer interfaces controlled by the Aspect Moderator using a vertical branch of the framework as illustrated in Figure 1 on the right.

Layered Aspect-Oriented Systems: Implementation

In [4] we introduced the Aspect-Moderator Framework (AMF). Components of the framework have well-defined roles and interactions:

- Each object (component) provides its services (methods) stripped of any aspectual properties (for example, no synchronization is included in Buffer objects).
- A proxy object intercepts called methods and transfers the calls to the Aspect Moderator.
- An Aspect Moderator object consists of the rules and strategies to bind aspects at runtime. Aspects are selected from the AspectBank. The Aspect Moderator orders the execution of aspects. The order of execution can be static or dynamic. Then, each precondition will be checked whether it is satisfied or not.
- An AspectBank object consists of aspect objects that implement different policies of variety of aspects.

The Layered Aspect Moderator Frameworks (L-AMF) [8] extends and modifies the basic AMF adding an inter-layer control to enable the design of Open Layered Aspect-Oriented Systems. The AbstractFactory, the Adaptor, and Bridge patterns [6] are used to define abstraction and support dynamic binding of aspects. Upper layer components can select services, and alter the specific aspectual properties by choosing from the set of different policies, which are provided by the lower layer. It may also either add new policies to an existing aspect (e.g., add a new scheduling policy to an existing scheduling aspect) or create a new aspect (e.g., add a security aspect and its different policies). Reconfigurability is built into the software architecture. An important property of the framework is its closure under both intra-layer weaving of aspects and components and inter-layer reuse. A layer application framework (the horizontal branch) and the system framework (the vertical branch) are unified in one framework. A user can employ application framework for application software and a system framework for system software. This provides a structured discipline to resolve the information-hiding dilemma with respect to aspectual properties of systems.

Since we are using well-known patterns and the UML representation of our framework. UML case tools can be used to develop both application and system software. Automated code generation can produce target code for a specific system.

Performance

Software performance is application dependent. Each application might have different tradeoffs for performance and other software properties. Moreover, the same application may need to switch policies that can affect performance at runtime. By providing openness on both cross-layer reuse and inter-layer composition, each application can either alter aspectual properties from a predefined set or add an application specific one to achieve a desired balance. Quality of Service (QoS) may depend on lower level system implementation of specific aspects. By making these accessible and modifiable at the application level and using a structured discipline, we can make better choice between QoS policies. System software is often critical to performance and also critical to understanding system development. System performance normally requires tuning to the particular kinds of traffic and use that a given application requires. We have applied AOP [7] to this problem to build system software that can be tunable for each application. In fact, the mechanism is sufficiently dynamic to monitor its own behaviors and change algorithms on the fly, matching current needs.

By using a layered approach, we inherit both the advantages and disadvantages of this particular modularization of systems. One disadvantage is the overhead associated with inter-layer system calls. Open implementation also has its own overhead. For today complex systems that require evolving requirements and customized features an operating cost is unavoidable. A real life system may have to choose between tradeoffs.

Conclusion

We have developed an Open Layered Aspect-Oriented System Framework. The open L-AMF facilitates the separation of aspects, components, and layers and provides closure under composition rules. This closure under composition of aspects, components and layers gives the framework its desired applicability.

As object-oriented operating systems have emerged from OOP to provided object-orientation advantages for systems software, we believe that *Aspect-Oriented System* has the potential to support the next generation software systems. Such systems use aspect orientation to enhance comprehension, organization, and manipulation of system aspectual properties. Other applications of AOP to system software can be found in [2, 3].

With design patterns and language independence, software developers can use CASE tools, especially UML, to deploy Open L-AMF in their target languages and systems. An UML Aspect-Oriented Modeling to support our framework has been presented in [1].

The L-AMF has been implemented using C++ and tested with the classic Operating System benchmarks such as producer/consumer and reader/writer problems. We have tested system evolution by statically removing or adding new aspects such as mutual exclusion, logging, tracing, and fault tolerance. For each system evolution the Aspect Moderator made only localized modifications [8].

Further information can be found at http://www.iit.edu/~concur

References

[1] Aldawoud O., A. Bader, C. Constantinides, and T. Elrad. Modeling Intra object Aspectual Behavior. ICSE Workshop on Describing Software Architecture with UML. 23ed International Conference of Software Engineering. May 2001.

[2] Atkinson C. and T. Kühne. Separation of Concerns through Stratified Architecture. Workshop on Aspects and Dimensions of Concerns at ECOOP 2000, Cannes, France, June 2000.

[3] Coady Y., G. Kiczales, and M. Feeley. Exploring an Aspect-Oriented Approach to Operating System Code. Workshop on Advanced Separation of Concerns at OOPSLA 2000, Minneapolis, MN, October 2000.

[4] Constantinides C. A., A. Bader, T. Elrad, M. E. Fayad, and P. Netinant. Designing an Aspect-Oriented Framework in an Object-Oriented Environment, *ACM Computing Surveys*, Vol. 32, No. 1es, Article No. 41, March 2000.

[5] Dijkstra E. W. The Structure of THE Multiprogramming System. *Communications of ACM*, pp. 341-346, May 1968.

[6] Gamma E., R. Helm, R. Johnson, and J. Vlissides. *Design Pattern: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995.

[7] Kiczales G., J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. -M. Loingtier, and J. Irwin. Aspect-Oriented Programming. *ACM Computing Surveys*, Vol. 28, No. 4es, Articles No. 154, December 1996.

[8] Netinant P., C. A. Constantinides, T. Elrad, and M. E. Fayad. Supporting Aspectual Decomposition in the Design of Adaptable Operating Systems Using Aspect-Oriented Frameworks. *Proceedings of 3rd Workshop on Object-Orientation and Operating Systems ECOOP 2000*, pp. 36-46, Sophia, France, June 2000.

[1] Bangkok University
Computer Science Department
Bangkok, Thailand.

[2] Concurrent Programming Research Group

Computer Science Department
Illinois Institute of Technology
Chicago, Illinois.

[3]Department of Computer Science and Engineering
University of Nebraska at Lincoln
Lincoln, Nebraska.