

# Adding protection to the object model as the basis for secure operating systems

Darío Álvarez Gutiérrez, María Ángeles Díaz Fondón, Iván Suárez Rodríguez,  
Fernando Álvarez García, Lourdes Tajés Martínez  
Dept. of Informatics, University of Oviedo  
c/ Calvo Sotelo s/n 33007, Oviedo, Spain  
darioa@uniovi.es, fondon@uniovi.es, banisr@telecable.es,  
falvarez@uniovi.es, tajes@uniovi.es

## ABSTRACT

Operating Environments (Operating Systems) are increasingly being built with object-oriented languages, for with virtual-machine runtime environments implementing the object model of the language. Adding protection to the traditional properties of the object model (the ability to protect a method call from an object to another object) can be of great advantage for building secure operating systems, using the protection property of the object model as the access-control mechanism of the system. We have used capabilities as the protection mechanism, implementing this approach into a system of our own, and a commercial system (Microsoft .NET, in the SSCLI version) with promising results.

## 1. INTRODUCTION: THE OBJECT MODEL, PROTECTION, AND OPERATING SYSTEMS

Operating Environments (Operating Systems) are increasingly being built with object-oriented languages. There is also a tendency to deliver the implementation using a virtual-machine runtime environment that implements the object model of the language (or a common object model for many languages). The services offered by the environment (that can be seen as including services of an Operating System) are constructed upon this basis, usually in the form of class libraries. Java and .NET are "commercial" examples of this approach. In general, this approach can also be used to build more "conventional" operating systems (in the sense of the abstractions offered).

The object model used (and implemented in the virtual machine) usually has these properties (taken from our own system [Dia99], but mostly based on the "classic" properties of an object model from Booch [Boo94]):

- Unique object identity (used by references)
- Encapsulation (object access only through methods)
- Classes (which are also used to derive types)
- Multiple inheritance (is-a) relationships
- Aggregation (is-part-of) relationships
- General association (related-to) relationships
- Polymorphism and type checking including run-time checking
- Exceptions
- Concurrency
- Persistence
- Distribution

A protection mechanism (access control mechanism) is a security measure in computer systems that restricts access from a piece of code (subject) to a resource (object). An example is the well-known Access Control List protection mechanism: variants are used in operating systems such as Unix.

In terms of OO systems, the subject is an object instance (client object), and the resource is another object instance (server object). Protection is then expressed as the ability to decide whether a method call from the client object to the server object should proceed or not.

We think that *protection* is important enough for future systems as to be considered part of the properties of the object model. Protection should not be realized outside of the core features of the object model, thus having to be built in an upper layer of the system.

By adding protection to the object model of the system, the environments built upon (operating systems) can just take it as the basis of the security measures provided in the system. In particular, object-model protection can be directly used as the protection mechanism (access control mechanism of the system).

## 2. USING CAPABILITIES AS THE PROTECTION MECHANISM OF THE OBJECT MODEL

To implement protection into the object model, a specific protection system should be chosen. We think capability-based protection is well-suited for these purposes, as it can be smoothly integrated with the object model.

Pure Capabilities [Den66] are a well-known protection mechanism, that can be used to implement a comprehensive set of flexible security policies. A capability is basically a ticket which names an object (resource) and a set of permitted operations on that object (permissions) (Figure 1). The only requirement for an object (subject or client) in order to use another object (object or server) is to hold a capability pointing to the server object with adequate permission to use the intended operation. Consequently, an object will hold just the minimum protection information relevant to it: the rights to just the objects it will use.

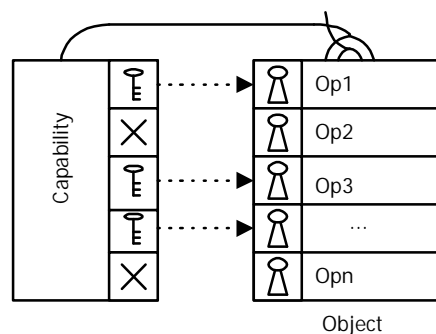
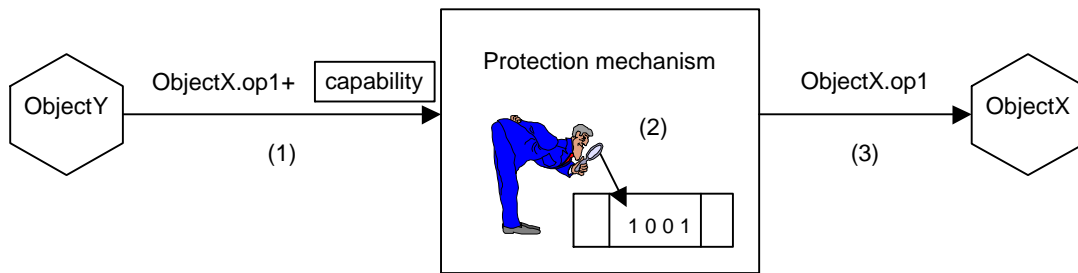


Figure 1. A capability.

Conceptually, the protection information (permissions) can be integrated with object references in the virtual machine implementing the object model. In order to test whether the method call from a client to a server object should proceed, the model must check whether the client object holds a permission for the method called within the reference to the server object. If the reference does not hold a permission for the method called in the destination object, the call fails, and an exception is raised (Figure 2).



**Figure 2. Checking permissions in a capability.**

The operation of the system is not affected, but for the incremental property of protection added to the model. Basically, the creator of an object gets a reference with all the permissions set (the “owner” can call all the methods). Just one instruction is added to the abstract machine: the ability to restrict a permission in a reference. This allows to follow the principle of least privilege. The rest of the system is not affected: references are copied, passed as arguments, etc. in the usual OO way. However, references now carry permissions, and that is how privileges are acquired.

Capabilities have some drawbacks, most notably revocation problems, although solutions such as facades in various incarnations [Mil03] and reference monitors [Lan81] can be added if needed.

### 3. BENEFITS OF THE APPROACH

There are a number of benefits of this approach. On the one hand, this helps satisfying the classical Saltzer and Schroeder design principles for protection mechanisms [Sal75], in particular:

- **Simplicity (Economy of Mechanism).** Complicated protection mechanisms that rely on interaction of many elements are prone to security holes. Here, the protection mechanism is an integral part of the system, not an afterthought, conceptually simple, and small enough to be easily verified.
- **Complete Mediation.** The protection mechanism is located at the innermost level of the system (references of the abstract machine and method calls), being virtually impossible to overcome. As all operations are expressed as method calls, all operations in the system are checked (mediated), so possible security holes are minimized.
- **Fine granularity of protection (Least Privilege).** As seen, individual operations (methods) can be protected, for any object in the system. So there are no restrictions to accomplish the principle of least privilege.

There are also a number of advantages more directly related to capability-based protection itself:

- **Automatic protection of capabilities.** Advantages of both segregated and user (sparse) capabilities are combined. Tampering of capabilities is not possible as the virtual machine prohibits this, because only the operations provided by the machine can be used to handle capabilities. However, capabilities can be flexibly used in normal user structures (still being used as normal references).

- **Arbitrarily long permissions.** Many systems present capabilities as a fixed string of bits holding permissions for operations [Tan86]. So only predefined or a fixed number of operations can be protected. Full object semantics require each and every method of an object to be protected. Capabilities are presented as a high level abstraction that hides their internal structure of capabilities, so implementation can be carried the way that seems most adequate. Hence, permissions can be arbitrarily long, protecting all operations on an object.
- **OO uniformity.** Protection is integrated into the object model of the system in a uniform way. There are only minor changes to the object model and to the behaviour of the system. Only one new instruction is just needed for protection (an instruction to restrict the permissions on a reference). The protection mechanism is transparent for the rest of the system. For example, existing applications will work without any modification exactly as before when applied to existing virtual machines with capabilities enabled (but, of course, can not take advantage of capabilities without modifications that use the new support for capabilities).

#### 4. IMPLEMENTATION IN EXPERIMENTAL AND COMMERCIAL SYSTEMS

We have implemented this approach in two systems based on object-oriented abstract machines.

The first one was our own experimental system Oviedo3 [Dia99]. The performance penalty of capability-based protection was about 0,265% per protected method calls. As many method calls are to local objects that are not shared (i.e. all permissions are set in a reference held by the creator of the object), and thus the call will never be rejected, this case was optimized.

We also implanted capability-based protection into the SSCLI-Rotor (the RotorCapa system<sup>1</sup>). SSCLI-Rotor [Stu03] is Microsoft's Shared Source implementation of the Common Language Infrastructure (.NET). Here the performance was at least equal to the stack-introspection protection mechanism shipped with .NET, and even better depending on the stack size.

#### 5. RELATED WORK

Capabilities as a protection mechanism are almost as old as computers, and many projects have used this protection mechanism, especially in the OS area. With the recent spread of security breaches in commercial Access Control List-based operating systems, there is a renewed interest in them, as shown in the EROS operating system [Har02], or the JX [Wav03] operating system.

Capabilities were also used in object-oriented systems, for example in the Hidden Capabilities model [Hag96]. They were also used in OO systems based in Virtual Machines, such as the J-Kernel [Haw98] project for the Java platform, and the already mentioned JX OS. These projects have in common the use of the basic philosophy of capabilities for protection. However, our approach differs mainly in how protection is smoothly integrated with the references and method calls of the object model and with the virtual machine structures and mechanisms.

---

<sup>1</sup> RotorCapa development was supported by Microsoft Research through the RFP Rotor Awards. Home page is at <http://www.di.uniovi.es/~darioa/rotorcapa/> and at the SSCLI Community Site <http://rotorcapa.sscli.net/>

## 6. REFERENCES

- [Boo94] Booch, G. Object-Oriented Analysis and Design with Applications, 2<sup>nd</sup>. ed. Benjamin Cummings, 1994.
- [Den66] Dennis, J., and van Horn, E. Programming Semantics for Multiprogrammed Computations. *Comm. of ACM* V9,3, 1966.
- [Dia99] Díaz Fondón, M.A., Álvarez Gutiérrez, D., García-Mendoza Sánchez, A., Álvarez García, F., Tajés Martínez, L., and Cueva Lovelle, J.M. Integrating Capabilities into the Object Model to Protect Distributed Object Systems. *International Symposium on Distributed Objects and Applications (DOA'99)*, IEEE Computer Society Press, pp. 374-383, 1999.
- [Hag96] Hagimont, D., Mosière, J., Rousset de Pina, X., and Saunier, F. Hidden Capabilities. *16th International Conference on Distributed Computing Systems*, May 1996.
- [Har02] Hardy, N., and Shapiro, J. EROS: A Principle-Driven Operating System from the Ground Up. *IEEE Software*, pp 26-33, January 2002.
- [Haw98] Hawblitzel, C., Chang, C., Czajkowski, G., Hu, D., and von Eicken, T. Implementing Multiple Protection Domains in Java. *1998 Usenix Ann. Tech. Conf.*, Louisiana, June 1998.
- [Lan81] Landwehr, C.E., *Formal Models for Computer Security*. *ACM Computing Surveys*, vol. 13, no. 3, September 1981.
- [Mil03] Miller, M.S, and Shapiro, J.S. Paradigm Regained: Abstraction Mechanisms for Access Control. *8th Asian Computing Science Conference (ASIAN03)*, pp. 224-242, December 2003.
- [Sal75] Saltzer, J.H., and Schroeder, M.D. The Protection of Information in Computer Systems. *Proceedings of the IEEE.*, vol. 63, no. 9., pp. 1278-1308, September 1975.
- [Stu03] Stutz, D., Neward, T., and Shilling, G. *Shared Source CLI Essentials*. O'Reilly, 2003.
- [Tan86] Tanenbaum, A.S., Mullender, S.J., and van Renesse, R.. Using Sparse Capabilities in a Distributed Operating System. *6th Int. Conference on Distributed Computing Systems*, 1986.
- [Wav03] Wawersich, C., Felser, M., Golm, M., and Kleinöder, J. The Security Architecture of the Java Operating System JX - A Security Architecture for Distributed Parallel Computing. *5th Advanced Parallel Programming Technologies International Workshop (APPT 2003)*, pp. 85-95, September 2003,