Why Capability-Based System?

Nur Izura Udzir

March 22, 2007

1 Loose Control in TS-Based Systems

Coordination in distributed computing involves a group of loosely coupled agents cooperating in order to solve problems in a decentralized fashion. One of the popular approaches to coordination is generative tuple-space based communication, where agents interact via a shared data space.

LINDA, the shared data space coordination paradigm, is a popular alternative to the conventional point-to-point communication approaches, and its advantages have been described in earlier chapter.

Despite LINDA's powerfulness, it is unfortunate, however, that its very reliance on open and flexible communications leaves it vulnerable to manipulations, which gives rise to a lot of relevant security problems, like secrecy and integrity of data and program code. This becomes particularly problematic in the case of coordinating multiple intelligent, and mostly mobile, agents.

This shortcomings has led to the exploitation of several alternative approaches to enforce security policies in distributed computing systems, and to marry them with tuple-space based coordination in the hope of having a system that offers secure conversation while enjoying the flexibility and openness of the tuple-space models. These 'secure' variants of LINDA assist greatly in alleviating the security problems, but do not fully control every aspect of coordination that ought to be controlled.

Capability-based coordination may offer a solution to the problem. Acting as a 'ticket', capabilities provide a means to control agents' access to objects in the system. These tickets can be given to the chosen agents granting them different privileges over different kinds of data.

2 Security Issues

Throughout this thesis, it is assumed that capabilities are protected by means of some kind of security technique. Although the security aspect of the capability-based system is not the focus of this research, this does not diminish its importance. For protection, it is vital

that capabilities are unforgeable. There are a number of possibilities to make capabilities unforgeable:

- Hardware tags: each word will be associated with a 1-bit tag [Ber80, Sol97]. The word containing a capability can be tagged "on" to prevent it from being modified or copied by agents. This scheme has yet to be proven a successful implementation. The research reported in this thesis, however, does not explore this kind of low-level mechanism.

- Protected address space: capabilities are stored in a memory area that is not accessible to users. The protection relies on the fact that capabilities are not allowed to migrate into any address space directly accessible by the users (where they could be modified) [PS85]. This scheme requires the capabilities to be maintained in a centralized list (known as capability list or c-list), which is not favourable to distributed environment.

- Language-based security: using programming languages to enforce protection on capabilities, an approach adopted in Java, for example. This scheme, however, is not suitable for open heterogeneous systems, as the coordination and computation languages need to be orthogonal.

- Cryptography: capabilities are encrypted, and can only be decrypted using the designated key(s). This is possibly the most popular scheme. For example, VLOS [CM02b] used the RipeMD-128 cryptographic hashing algorithm, a variation of the RipeMD-160 cryptographic hashing function [DBP96], to calculate its capabilities' 128-bit hash value. Amoeba [TMvR86] uses both the conventional and public-key encryption to protect its capabilities from being forged. Capabilities in any message are encrypted with a unique key made up from the source machine and the destination machine identifiers. Another tuple-space based system, the Tagged Sets [OH05] use public- and private-key encryption mechanism to protect their shared tagged sets in a distributed environment.

Digital signatures are often used for message authentication—messages are digitally signed with the sender's private key, and consequently the sender's public key can be used to check the message's authenticity and that its contents are intact. Vitek *et al.* [VBO03] discusses the use of digital signature in SecOS where messages are appended with an extra field (the signature) which can be extracted by the receiver to be matched against the message. Keys for each field are inversed to produce a matching replica to the message, and the (signature) value is tied to the message. These are done using built-in functions which are not made directly available to untrusted agents—to prevent any misuse of the signature, or unauthorized template-keys constructions.

Even though capabilities in general are mainly associated with protection, the issue of security is not addressed in this paper. Rather, the discussion focuses on incorporating capabilities as a controlling mechanism, while assuming that the security aspect has been

dealt with using some kind of techniques, for example cryptography. If such security exists, then capabilities in this system can be made secure. If capabilities are secure, the objects they protect are also secure of unauthorized access.

3 Why Capabilities?

Capabilities [Dv66] have been used as the basis for a variety of systems (see [Lev84] for numerous examples). They have the attractive features of providing a single, uniform mechanism for naming, accessing, and protecting objects within the system. In all of these systems, the capabilities are managed by (trusted) software kernel. A capability [Nut02] is an unforgeable 'ticket' given to an agent that specifies which kind of operations on a certain object are permitted to the holder of the capability. Capabilities can be defined in a more general way: as 'visibility' filters to create a more refined control over agents' operations on objects in the system.

Capabilities are well suited for open distributed systems as they themselves are *distributed* in the sense that:

- the controlling attributes are held by the agents, rather than being attached to objects, thus putting no storage overhead on the objects.
- verification is made by the kernel upon the presentation of the capability by the agent, without the need to search any list. Therefore, verification time is constant in capability-based systems.
- the decision to grant a capability to an agent is the responsibility of some holder of the capability, not the kernel. The kernel only generates and checks the capabilities.
- capabilities can be transferred from one agent to another. This is a form of 'distribution' as, subject to certain constraints, any agent (not necessarily the object creator) possessing the capability can pass a copy of it to another agent.

In addition, a capability mechanism also supports the *flexibility* inherent in distributed systems:

- it accommodates user-defined rights, not restricted to those fixed by the system, thus allowing them to be dynamically changed; and
- its 'domain flexibility' feature allows agents to join and leave the system simply by requesting (and possessing) appropriate capabilities to access the objects, as opposed to having to modify numerous lists attached to each object relevant to the agents' execution.

3.1 Capabilities *versus* access control lists

An ACL is associated with each data object and consists of a list of users, enumerating what accesses to the object each user is permitted to exercise. ACLs can be difficult to administer. Expressing authorization for a large number of users becomes awkward when it entails managing lists comprising large numbers of entries. UNIX systems therefore employ a modified scheme: for each object, the owner only specifies object access permissions for the user, for a small number of specified groups of users, and for all other users. Windows NT also addresses this administration problem by supporting access permissions for groups.

The decision subsystem for an ACL-based discretionary policy simply obtains the name of the user on whose behalf a particular process is executing, checks the ACL for an entry containing that user name, and grants accesses according to the ACL entry that is found. This has been called a list-oriented approach.

An alternative to ACLs is to associate with each process a list of capabilities, each of which names an object along with the kinds of access to that object that the capability-holder is permitted (Kain and Landwehr, 1986). The decision subsystem for a capability-based access control mechanism checks the list of capabilities associated with the process making the access to see if a capability is present for the desired data object and access mode. This has been called a ticket-oriented approach.

As mentioned earlier, capabilities have attractive features that make them a suitable controlling mechanism for open, distributed and heterogeneous systems. Capability-based security mechanisms are more flexible than their ACL-based counterparts.

Access Control Lists (ACLs). Access rights are attached to the objects. These rights are represented by a list of sets of attributes, where each set of attributes represents the access permission granted to a certain domain of agents, e.g. the owner and all others.

This mechanism is favourable in many simple applications, because it corresponds directly to the needs of the users, allowing them to specify (upon object creation) which domain, and what kind of operations are allowed on it [PS85]. Unfortunately, in more complex situations, it is not very easy to implement: this method requires every access to the object to be checked, i.e. searching the access list—this can be time consuming in large systems.

Capabilities. Access rights are held by the members of a domain (the agents), instead of being attached to the objects. These rights must be obtained (by submitting a request) by the agents prior to performing any operation on the objects. If the request is granted, an unforgeable 'ticket' is returned to the requesting agent. The ticket (also known as 'capability') identifies the domain, the object, and the rights granted.

Although capabilities do not correspond directly to the needs of the users, they are useful for localizing information for a particular domain. Moreover, no list needs to be searched—the protection system needs only verify that the access is valid based

on the capability (for that access) presented by the process. Managing capabilities, however, may be quite inefficient, particularly in terms of revocation and controlling the propagation of capabilities.[1]

Capabilities can only refer to named objects, e.g. tuple-spaces, but not tuples or tuple elements, as they are nameless. Access control lists, on the other hand, have this advantage over capabilities in the sense that they are applicable to named as well as unnamed objects. However, they are not very practical in an open system due to its requirement of attaching large lists (of sets of system-fixed attributes) to each objects. Combined with the required distributed domain management, this inherently limits the flexibility and increases the complexity of the system.

Using capabilities, on the other hand, does not incur the abovementioned limitations, for the simple fact that:

1. Each agent stores its own set of capabilities, putting no overhead on the objects to which they refer. This is in contrast to some popular practice [PS85, TMvR86, CM02a, RH97] where the *kernel* needs to maintain a list which stores all capabilities in the system, thus defeating the purpose of using capabilities for a decentralized control in distributed systems.

2. The information contained in the capability itself is sufficient for the kernel to assess its validity of the operation(s) for the specified object.

3. Verification time is constant—this can prove to be advantageous in cases of large number of tasks having restricted access to an object [CM02b], as no list need be searched.

4. Granting capabilities for an object is the sole responsibility of its owner, leaving the kernel only with the tasks of generating and checking the capabilities.

5. It does not restrict the rights to be only of those fixed by the system, allowing them to be dynamically changed (which is essential in an open, and possibly a long life-span, system). Therefore, it accommodates user-defined rights—a kernel needs only to know that an operation exists in order for it to be able to grant, and verify, a right for that operation [Woo99].

6. Capabilities feature 'domain flexibility', which is lacking in access control lists—the domains in access control lists are fixed. The simple possession of a valid capability for an object authorizes the agent to access it [PS85], without requiring the concept of a user to be included in the kernel [CM02b].

7. Unlike access control lists, capabilities can be transferred from one agent to another. Although there needs to be some kind of control over the propagation of rights,[2]

[1] See Revocation on page **??** and Transitivity (for propagation of capabilities) on page **??** in Section **??**.

[2] In some applications, the distribution of capabilities may need to be restricted so as to prevent them from being freely-propagated, for more control.

this property adds more flexibility to the system. A capability transfer is rather like adding an agent to an access control list domain: a specific capability defines an (ACL) domain as consisting of those agents who (currently) hold the capability. This shows that capabilities support dynamically created domains.

8. Only a small overhead in extra communication costs are incurred in order to add capabilities to the other parameters in an operation [Woo99].

Despite the advantages of implementing capabilities, this approach, however, is more complicated compared to access control lists. Nevertheless, due to its suitability for open distributed systems, and partly inspired by this challenge, this area of research is pursued to obtain a more refined control in such an environment.

4 Why Capability-Based Coordination?

LINDA is a powerful model suitable for coordination in open, heterogeneous systems, but it has some drawbacks, particularly in the security and control aspects, inspiring much research work aimed at improving this aspect of LINDA. As mentioned in the previous chapter, capabilities have been incorporated in a number of tuple-space based systems [ML95, DFP98, GP03, CM02b, BRP02] and discussed in [Woo99]. In these systems, however, capabilities are mainly discussed as an access control mechanism.

The research presented in this thesis defines capabilities in a more general terms: as 'visibility' filters to create a more refined control over agents' operations on objects in the system. With capabilities, the (open) system is more finely controlled, yet still flexible—a strong enough reason for using capabilities in tuple-space systems. Although capabilities can only be applied to named objects, and tuples are nameless, this problem is overcome with the use of multicapabilities—a new concept introduced in this research.

Although capabilities have their advantages, they also have limitations, mainly related to their management, which deter their advancement (in terms of implementability) in large systems. Nonetheless, the flexibility and distributive properties of capabilities make it worth exploring the possibility of incorporating them as the controlling mechanism in the tuple-space coordination systems with the aim of obtaining a more refined control in open, heterogeneous environments.

5 Summary

Capabilities possess the attractive features to act as a distributed controlling mechanism suitable for coordination of agents in open systems. To complement LINDA's powerful coordination properties, this thesis deployed capability-based control to overcome LINDA's lack of it. The following chapter will discuss the LINDA-like capability-based coordination system developed for this purpose.

References

[Ber80] Viktors Berstis. Security and protection of data in the ibm system/38. In *ISCA'80: Proceedings of the 7th Annual Symposium on Computer Architecture*, pages 245–252, New York, NY, USA, 1980. ACM Press.

[BRP02] Ciarán Bryce, Chrislain Razafimahefa, and Michel Pawlak. Lana: An approach to programming autonomous systems. In Boris Magnusson, editor, *ECOOP 2002, Lecture Notes in Computer Science 2374*, pages 281–308. Springer-Verlag, Berlin Hiedelberg, 2002.

[CM02a] Voon-Li Chung and Chris S. McDonald. The continuing evolution of VLOS. In Hamid R. Arabnia, editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA '02, Las Vegas, Nevada, USA, Volume 3*, pages 1385–1392. CSREA Press, 24-27 June 2002.

[CM02b] Voon-Li Chung and Chris S. McDonald. The development of a distributed capability system for VLOS. *Australian Computer Science Communications*, 24(3):57–64, Jan-Feb 2002. Appeared in Proceedings of the 7th Asia-Pacific Conference on Computer Systems Architecture - Volume 6, Conferences in Research and Practice in Information Technology Series.

[DBP96] Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. RIPEMD-160, a strengthened version of RIPEMD. In D. Gollmann, editor, *Fast Software Encryption, Lecture Notes in Computer Science 1039*, pages 71–82. Springer-Verlag, Berlin Hiedelberg, 1996.

[DFP98] Rocco De Nicola, Gian Luigi Ferrari, and Rosario Pugliese. KLAIM: A kernel language for agents interaction and mobility. *IEEE Trans. on Software Engineering*, 24(5):315–330, May 1998.

[Dv66] J. B. Dennis and E. C. van Horn. Programming semantics for multiprogrammed computations. *Communications of the ACM*, 9(3):143–155, March 1966.

[GP03] Daniele Gorla and Rosario Pugliese. Enforcing security policies via types. In D. Hutter, G. Mueller, W.Stephan, and M. Ullman, editors, *Proc. of 1st International Conference on Security in Pervasive Computing (SPC'03), Lecture Notes in Computer Science 2802*, pages 88–103. Springer-Verlag, 2003.

[Lev84] Henry M. Levy. *Capability-Based Computer Systems*. Digital Press, 1984. http://www.cs.washington.edu/homes/levy/capabook/.

[ML95] Naftaly H. Minsky and Jerrold Leichter. Law Governed Linda as a coordination model. In Paolo Ciancarini, Oscar Nierstrasz, and Akinori Yonezawa, editors, *Object-Based Models and Languages for Concurrent Systems, Lecture Notes*

in Computer Science 924, pages 125–146. Springer-Verlag, Berlin Hiedelberg, 1995.

[Nut02] Gary Nutt. *Operating System: A Modern Perspective*. Addison-Wesley, USA, 2 edition, 2002.

[OH05] Manuel Oriol and Michael Hicks. Tagged sets: A secure and transparent coordination medium. In Jean-Marie Jacquet and Gian Pietro Picco, editors, *Proceedings of the Seventh International Conference on Coordination Models and Languages, Lecture Notes in Computer Science 3454*, pages 252–267. Springer-Verlag, Berlin Hiedelberg, April 2005.

[PS85] James L. Peterson and Abraham Silberschatz. *Operating System Concepts*. Addison-Wesley, USA, 2 edition, 1985.

[RH97] Thomas Riechmann and Franz J. Hauck. Meta objects for access control: Extending capability-based security. In *Proc. ACM New Security Paradigms Workshop 97 (NSPW97)*, Great Langdale, UK, 1997.

[Sol97] Frank G. Soltis. *Inside the AS/400, Featuring the AS/400e series*. 29th Street Press, Loveland, CO, USA, 1997.

[TMvR86] Andrew S. Tanenbaum, S. J. Mullender, and R. van Renesse. Using sparse capabilities in a distributed operating system. In *Proceedings of the 6th International Conference on Distributed Computing Systems (ICDCS)*, pages 558–563, Washington DC., 1986. IEEE Computer Society.

[VBO03] Jan Vitek, Ciarán Bryce, and Manuel Oriol. Coordinating processes with secure spaces. *Science of Computer Programming*, 46(1-2):163–193, 2003.

[Woo99] Alan Wood. Coordination with attributes. In Paolo Ciancarini and Alexander L. Wolf, editors, *Coordination Languages and Models: Proc. 3rd International Conference COORDINATION'99, Lecture Notes in Computer Science 1594*, pages 21–36. Springer-Verlag, Berlin Hiedelberg, 1999.