Fragmentation of Confidential Objects for Data Processing Security in Distributed Systems*

J.-C. Fabre T. Pérennou

LAAS-CNRS & INRIA
7, avenue du Colonel Roche
31077 Toulouse, France

Abstract

This paper discusses how object orientation in application design enables confidentiality aspects to be handled more easily than in conventional approaches. The idea, based on object fragmentation at design time, is to reduce processing in confidential objects; the more non confidential objects can be produced at design-time, the more application objects can be processed on untrusted shared computers. Still confidential objects must be processed on non shared trusted workstations. Rules and limits of object fragmentation are discussed together with some criteria evaluating trade-offs between fragmentation and performance.

1 Introduction and problem statement

Processing confidential information in an hostile environment is a difficult issue that has been previously tackled through conventional solutions. A first solution is to process ciphered information [1]; this solution is mainly based on special ciphering techniques called privacy homomorphisms but, although attractive, it is limited in use and can be subject to simple intrusions as described in [2]. A much simpler solution, very costly and of course not realistic, is to process clear information on trusted and physically protected non-shared computers. However, in today's systems architecture, one should take benefits of powerful shared heterogeneous computers, some of which being specialized for a given part of the application, without making strong assumptions about their internal security and surrounding environment.

However, distribution has been initially seen as a conflicting paradigm as far as confidentiality is concerned. Actual solutions to distributed security rely on protection mechanisms (notion of trusted computing base [3]); this means that part of the system must be trusted.

Nevertheless, an efficient protection of confidential information in a distributed architecture may be difficult and sometimes insufficient with standard computers and operating systems. Actually, applications *should* take advantage of the distributed architecture and its specialized components without endangering confidentiality although the information is processed in clear. The main challenge of the work reported in this paper is to propose a new approach for designing such applications.

The proposed approach is part of a general technique for handling both accidental and intentional faults in distributed systems, called Fragmentation-Redundancy-Scattering (FRS) [4, 5]. We concentrate here on its application to confidential information processing where the core aspect is fragmentation at design time.

An application can be organized in such a way that only a minimal part of the application needs a trusted execution environment. According to well defined assumptions and few mechanisms (section 2), the rest of the application can be executed on untrusted shared computers. This part of the application can thus be replicated for fault tolerance[1] and scattered without endangering confidentiality.

The approach consists first in identifying at the design phase the information which is confidential. Then, the application can be divided into two parts: confidential data processing and non confidential data processing (section 3). Minimizing confidential data processing is the main objective of the *fragmentation process* which is described in this paper. Using an object-oriented design, the application is seen at each design iteration as a collection of objects. The re-

*This work has been partially supported by the ESPRIT Basic Research Action n°6362, PDCS2 (Predictably Dependable Computing Systems).

[1] Whatever the type of fault, physical fault or sensitive information destruction.

0-8186-7125-4/95 $04.00 © 1995 IEEE

Authorized licensed use limited to: ISAE. Downloaded on June 8, 2009 at 04:34 from IEEE Xplore. Restrictions apply.

sult of the fragmentation process is a new collection of objects, in which ideally few objects are confidential and must be executed on trusted non-shared computers (section 4). The amount of costly non-shared trusted resources is minimized, thus allowing more untrusted shared computers to be used. The method is very application-dependent and of course in some situations the result may be not satisfactory because of performance overheads. Finally, quantitative and qualitative aspects of performance evaluation are discussed (section 5).

2 System environment and related assumptions

2.1 System architecture

The architecture of the system (see figure 1) is composed of trusted user workstations and untrusted processing servers. User workstations are non shared devices[2] with weak computing and memory resources whereas processing servers are shared computing resources where different objects belonging to various applications can be executed simultaneously in an efficient way. The user workstations are trusted computing resources in such a way that, according to their protection mechanisms and surrounding physical environment, the probability of an intrusion during a user session is very low. On the contrary, shared processing servers can be subject to non restrictive intrusions (passive, active, malicious attacks on memory segments, system and temporary files, etc., even performed by privileged users, namely host administrators). These shared processing servers can be various specialized off-the-shelf computers with standard operating systems. Confidential information is limited and only processed on few non shared trusted workstations.

Any user application is only activated from its trusted user workstation and may involve using untrusted processing servers for running parts of the application.

2.2 Software architecture and system services

Applications will be seen at runtime as a collection of non shared distributed runtime units (active objects) interacting by messages. The underlying runtime system should enable active objects to be executed on any processing unit and to communicate with each other. The ideal runtime support can be an object oriented runtime layer, such as COOL [6]).

[2] They also can be time shared devices: only one user session at a time and no remote access during the user session at any abstraction level.

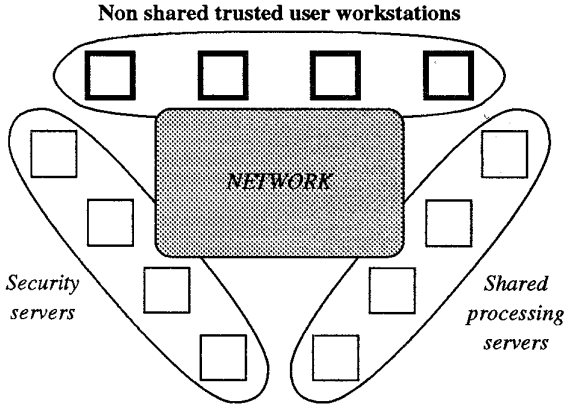

Figure 1: System architecture and services

We also suppose that user authentication, authorisation mechanisms and key management are performed by security services running on security servers as in [4, 7] (see figure 1). When the application is activated by an authenticated user, some active objects are loaded on the trusted user workstation (confidential objects) and other on untrusted processing servers (non confidential objects). Each active object is uniquely identified by a reference. The set of objects (references) belonging to a single application is only known at the trusted location. Active objects located on untrusted servers do not know each other even if they belong to the same application. We then suppose that intruders on processing servers are not able to identify objects belonging to a given application. Any object invocation is protected by authentication mechanisms (using public key authentication schemes for instance, e.g. [8]) thus preventing illegal invocation by intruders.

3 Confidentiality and object orientation

3.1 Confidentiality in applications

The notion of confidential information relates to the interpretation an intruder can have about its semantics in a given operational context. Information semantics may be confidential depending on its value: for instance, a string of characters might be sufficiently meaningful in isolation to be easily interpreted as a confidential information independently of any usage in a program. But this is not always the case; a numerical value is most unlikely to be interpreted as a confidential information without any knowledge of its internal representation or of its usage in a given application context. For example, a real variable is a confidential salary information if and only if it is associated

Authorized licensed use limited to: ISAE. Downloaded on June 8, 2009 at 04:34 from IEEE Xplore. Restrictions apply.

to a given person, period and currency. However, this viewpoint is also true using a very coarse granularity; for instance, let us consider a medical record system where the information is classified into two parts, administrative and properly medical. In this quite simple example, confidentiality is preserved as soon as the link between these two large fragments (some references) is retained at the trusted site.

In these simple examples, we can see that structuring a confidential information enables such information to be perceived as a set of non confidential items. The classical approaches do not take into account any structuring of the confidential information, often considered as a string of bits. Our approach relies on a different viewpoint: at a given abstraction level in the design of an application, most of the confidential data processed can be perceived as a collection of insignificant data items — only the links between such items reveals sensitive information to a potential intruder. When the information is not structured (e.g. strings, Unix files) confidentiality has to be maintained through classical solutions such as ciphering techniques, threshold schemes [9], IDA [10] and FRS [4].

3.2 Objet model and confidentiality

Designing the application as a set of objects enables confidentiality to be precisely identified in the application, since, from a software engineering viewpoint, objects represent real abstractions whose semantics is well known (a person, a medical record, a bank account, a key). The confidentiality of an object is considered as a boolean function and thus at any level of abstraction the set of application objects can be easily divided into confidential (set $\mathcal{C}$) and non confidential objects (set $\mathcal{NC}$), as shown in figure 2.

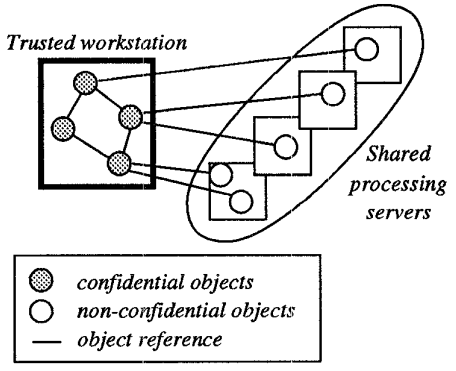

Figure 2: Confidential and non confidential sets of objects

The architecture of the application will then be composed of confidential objects that will be located on trusted workstations with references to non confidential objects located on shared processing servers; the non confidential objects do not communicate with each other, but only with one or several confidential objects.

In our approach, the identification of confidential objects starts from early stages in the design. Such confidential objects can be designed as a set of new objects, some of which not being confidential. This idea leads to extract as much as possible information and processing from the confidential objects. In figure 3 the object O has been re-designed as set of new objects $(O_1, O_2, O_3, O_4, O_5)$. The objective of this transformation is to have the amount of data and/or processing in the still confidential object O_1 and O_2 much lower than in O. Only objects that cannot be usefully substituted process confidential information.

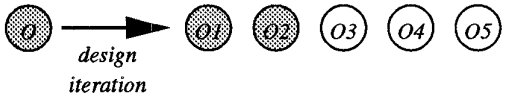

Figure 3: Confidential object transformation

This approach can be generalized to multi-level security: an application can be divided into secret objects (set $\mathcal{S}$), confidential objects (set $\mathcal{C}$) and non confidential objects (set $\mathcal{NC}$), objects with a high level of classification being re-designed as sets of objects with a lesser classification level.

4 Fragmentation

Fragmentation is an iterative process and provides the designer of a confidential application with a general framework which is in fact a variant of a classical object oriented design (in our case similar to a hierarchical object oriented design like HOOD). For the sake of simplicity, we consider here as a starting point of the fragmentation process that the application is composed of a unique confidential object which satisfies the functional specifications. At each step of the design process, the designer obtains a set of objects which satisfies the functional specifications of the application at some abstraction level. Confidential objects are identified within the current set. As stated in section 3, a confidential information can consist of related items including non-confidential ones. Such a structuring is used to design confidential objects as a set of (new) objects including non confidential ones. Among the objects thus produced, the confidential ones are examined at the next step. The

Authorized licensed use limited to: ISAE. Downloaded on June 8, 2009 at 04:34 from IEEE Xplore. Restrictions apply.

process stops either when no more object is confidential or when confidential objects are fine-grain or when no substitution of a confidential object is interesting with respect to confidentiality (e.g. when all objects produced are confidential).

Those points are developed throughout the rest of this section: accurate definitions are given and guidelines for the substitution of confidential objects are discussed.

4.1 Definitions

The following definitions will be used in the rest of this paper.

Objects. We consider here a simple object model, where an object encapsulates data and provides a set of operations to manipulate these data[3]. The interface of an object x is denoted $\text{intf}(x)$. Figure 4 provides an accurate graphical representation of an object which will be used later in this paper. Confidential objects will be presented in grey. Let $\mathcal{O}$ be the set of objects thus defined.

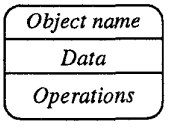

Figure 4: Extensive representation of an object

Confidentiality. As stated previously objects represent real abstractions, thus allowing the designer to decide whether an object is confidential or not according to the specifications. Let $C: \mathcal{O} \mapsto \{\text{TRUE}, \text{FALSE}\}$ be the predicate characterizing confidential objects.

Substitution mechanism. Substitution is a designer action that consists in replacing an object x by a non empty set of objects S_x such that S_x provides the same functionalities as x, which we denote by $S_x|x$ (this is read as S_x *substitutes for* x).

If a set of objects S_x substitutes for an object x, the services provided by x to the rest of the application must be distributed among the objects in S_x. The interface of x is then either located in a unique object in S_x or distributed among several objects in S_x. Throughout the rest of this paper we adopt the first solution which corresponds to a conventional object decomposition:

[3]This definition does not preclude considering an object as an instance of a hierarchy of classes from a software engineering viewpoint.

$$\forall x \in \mathcal{O},\ S_x|x \Leftrightarrow \begin{cases} S_x \subset \mathcal{O} \text{ cooperatively provides} \\ \qquad \text{the same services as } x \\ \exists x' \in S_x,\ \text{intf}(x) = \text{intf}(x') \end{cases}$$

Substitutable object. A confidential object can be usefully substituted for by a collection of objects when:

- non confidential new objects can be produced;
- gathering such non confidential objects does not enable the confidential object to be easily obtained;
- processing in non confidential objects is heavy enough to justify the substitution.

All these conditions must be satisfied to consider a confidential object as substitutable. For instance when dealing with fine-grain objects such as integers or strings the two last conditions are often unsatisfied. Such objects must then be ciphered using conventional techniques (e.g. strings) or be kept in the trusted area (e.g. integers). Let $S: \mathcal{O} \mapsto \{\text{TRUE}, \text{FALSE}\}$ be the predicate characterizing substitutable objects. In summary:

$$\forall x \in \mathcal{O},\ S(x) \Leftrightarrow \begin{cases} C(x) \\ x \text{ satisfies each of the three} \\ \qquad \text{preceding conditions} \end{cases}$$

4.2 More about the substitution mechanism

This section discusses how the designer should perform the substitution of a confidential object. From the confidentiality viewpoint, the interest of substitution is to produce non confidential objects. When identifying a confidential object, the designer should answer the following questions:

1. Why is this object confidential?
2. How can it be structured and how does it perform the provided operations?
3. Is this structuring suitable with respect to confidentiality and/or performance?

The first question should suggest part of the appropriate structuring requested by the second question, while the third one evaluates the usefulness of the substitution, particularly with respect to other solutions (including keeping the object as a whole, i.e. a still confidential object).

Authorized licensed use limited to: ISAE. Downloaded on June 8, 2009 at 04:34 from IEEE Xplore. Restrictions apply.

At early stages in the design, objects represent complex abstractions. In the example illustrated by figure 5, the designer identifies x as confidential and providing an operation called F. Then he determines that x is confidential because it includes a relation between two objects y and z, both providing operations G and H. So he decides to design x as an object x' holding two references to y and z and describes the algorithm of F in terms of y and z, and their operations G and H (see figure 5). This substitution is suitable because:

- the confidential association has been broken into separate items and non confidential ones have been identified (C1);
- gathering y and z does not provide any confidential information because the link between them is a complex algorithm F hidden in x' (C2);
- operations provided by y and z are considered CPU-time consuming (C3).

After the substitution the relation that makes x confidential is distributed among x', y and z, x' playing the role of a key; x' is therefore confidential because it holds the links allowing to rebuild the confidential information.

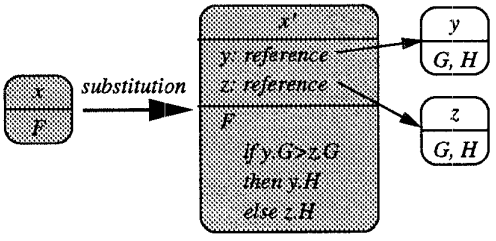

Figure 5: Early substitution

Later in the design, defined objects become closer to the implementation level of the corresponding abstraction. The structure of a confidential object should then appear to the designer as an aggregation of data of simple types within the confidential objects, the operations being series of simple instructions thus making the link rather weak with respect to confidentiality. Figure 6 illustrates this in terms of objects.

The structuring might seem appropriate since the '+' relating y and z remains hidden in x'. But if we look more carefully at the objects, we can see that y and z are in fact local to x' since $y = y.GET$ and $z = z.GET$. So this decomposition is totally useless with respect to confidentiality because x' still fully associates y and z as x did, an also because the operator

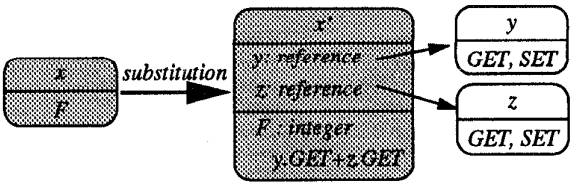

Figure 6: Late substitution

linking them is simple. Moreover operations provided by y and z are simple SET and GET operations.

So why not considering such x — and more generally objects dealing with fine-grain attributes — as non substitutable? Actually, the granularity of the objects produced is not sufficient to answer the question. It depends on the complexity of all the operations provided by y and z: if they are CPU-time consuming operations then it can be interesting to run them on untrusted shared resources.

Going through the fragmentation process obviously leads in most cases to fine-grain objects and performance evaluation is needed to state on the usefulness of a substitution. This issue will be discussed in section 5.

4.3 Formal description of fragmentation

The process consists in producing recursively a set of objects covering the functional and confidentiality aspects of the specifications. It will be presented here in an algorithmic form. At each step $i \geq 0$, $\mathcal{E}_i$ is the set of objects not treated yet. Each step of the algorithm can be described as follows.

4.3.1 Fragmentation algorithm

Let $\mathcal{E}_0 \subset \mathcal{O}$ be the set of objects deducted from the functional specifications. At each step $i \geq 0$, $\mathcal{E}_i$ is partitioned in confidential objects, constituting a set $\mathcal{C}_i$, and non-confidential objects, constituting the set $\mathcal{NC}_i$.

$$\begin{aligned} \mathcal{C}_i &= \{x \in \mathcal{E}_i \mid C(x)\} \\ \mathcal{NC}_i &= \{x \in \mathcal{E}_i \mid \neg C(x)\} \end{aligned}$$

Among the objects of $\mathcal{C}_i$, some can be substituted for and others cannot according to the criteria defined in section 4.1. $\mathcal{C}_i$ is therefore partitioned in $\mathcal{S}_i$ and $\mathcal{NS}_i$, defined as follows.

$$\begin{aligned} \mathcal{S}_i &= \{x \in \mathcal{C}_i \mid S(x)\} \\ \mathcal{NS}_i &= \{x \in \mathcal{C}_i \mid \neg S(x)\} \end{aligned}$$

We now consider only the following sets, which are a partition of $\mathcal{E}_i$. $\mathcal{S}_i$ is the set of elements of $\mathcal{E}_i$ that

Authorized licensed use limited to: ISAE. Downloaded on June 8, 2009 at 04:34 from IEEE Xplore. Restrictions apply.

are confidential and can be substituted soundly. $\mathcal{NS}_i$ is the set of elements of $\mathcal{E}_i$ that are confidential but cannot be substituted soundly. $\mathcal{NC}_i$ is the set of non-confidential elements of $\mathcal{E}_i$.

We substitute each element $x \in \mathcal{S}_i$ with a set $S_x \subset \mathcal{O}$, i.e. $S_x|x$, functionally equivalent, i.e. the interface and services provided by x are also provided by S_x. The algorithm then continues with $\mathcal{E}_{i+1}$ defined as:

$$\mathcal{E}_{i+1} = \bigcup_{x \in \mathcal{S}_i} S_x.$$

$\mathcal{E}_{i+1}$ holds all the objects substituted for all the confidential objects at step i: step $i+1$ will therefore only study new objects produced at step i.

4.3.2 Resulting sets and algorithm properties

A sufficient condition for termination is that no more object is confidential or can be substituted for, i.e. $\exists I \geq 0 \ \mathcal{S}_I = \emptyset$. Let then $\mathcal{E} = \bigcup_{i=0}^{i=I} \mathcal{E}_i \backslash \mathcal{S}_i$, $\mathcal{C} = \bigcup_{i=0}^{i=I} \mathcal{NS}_i$ and $\mathcal{NC} = \bigcup_{i=0}^{i=I} \mathcal{NC}_i$. $\mathcal{E}$ is the whole set of objects whose cooperation meets the application specifications, $\mathcal{C}$ is the subset of still confidential objects, and $\mathcal{NC}$ the set of non-confidential objects, both sets being a partition of $\mathcal{E}$.

This condition is in fact always satisfied since confidential objects granularity eventually becomes very small, and then by definition objects become no more substitutable since their interface comes down only to *SET/GET* operations. At one extreme, going down to a single bit, the criteria (C2) and (C3) are obviously not satisfied! More seriously conventional solutions must be used when confidential objects are simple types of unstructured data. This means that finally the application can include a large number of medium-grain or fine-grain objects[4] although large non confidential objects can be produced during the early stages of the design.

5 Performance evaluation

We have shown in the previous sections that a confidential application can be organized as a collection of objects, some of them being non confidential. For the sake of clarity, we consider here that the sets $\mathcal{C}$ and $\mathcal{NC}$ have been defined in a first step without considering performance aspects. Security is ensured as long as objects belonging to $\mathcal{C}$ are executed on the trusted user workstation (called PC i.e. some *personal computer* in the rest of this section) in a physically secure environment. The non confidential objects can then be executed on untrusted shared processing servers (called server in the rest of this section) without threatening the confidentiality of the whole application. We discuss in the following whether it is interesting or not from a performance viewpoint to remotely execute objects of $\mathcal{NC}$ on an untrusted server rather on the PC.

The performance evaluation of a fragmented application depends on several parameters related to both the application (size of invocation/reply messages, method execution time, objects implementation) and the system architecture (relative CPU power of PCs and processing servers, communication throughout the network). The organization of the confidential application in terms of distributed objects must respect some trade-offs between security (fragmentation), performance, cost and usage of the overall architecture. Remote execution of non confidential objects is interesting in several cases and for the non exclusive following reasons:

- when the communication overhead due to the cooperation of objects produced by the fragmentation process is balanced by the better performance obtained by executing non confidential objects on powerful remote computers (including computers with specialized architectures);
- when good parallelism is achieved among the non confidential objects executed remotely;
- when flexibility and extensibility of the network configuration and fair use of existing shared resources are also important goals;
- when fault tolerance is a prime objective that must be achieved through software-based fault tolerance techniques (e.g. non confidential objects can be replicated).

Only the first of these features can be evaluated quantitatively. However we believe that a qualitative evaluation of other remaining features is of great interest from a pure pragmatic viewpoint. These points are now discussed in detail.

5.1 Quantitative aspects

We suppose here in a first step that the architecture where the application is to execute is composed of a secure PC and powerful servers, e.g. 10 times as powerful as the PC. We discuss criteria for determining whether non confidential objects should be run on the PC or on the server and for evaluating the respective costs.

The execution of an object on the PC is almost immediate because it is not shared by several users; just few user applications are simultaneously active in

[4] It can also be noticed that the more objects become fine-grain, the more their semantics becomes difficult to obtain.

Authorized licensed use limited to: ISAE. Downloaded on June 8, 2009 at 04:34 from IEEE Xplore. Restrictions apply.

a time-sharing system[5]. On the contrary the remote execution of an object must include time spent on message passing and scheduling on a multi-user processing server.

Just for easy understanding, the balance can be illustrated by the simple following example; suppose that a method M of an object O is invoked by message *im* (invocation message) and results are returned by message *rm* (reply message). Table 1 summarizes the results of an experiment comparing a local vs. a remote execution of M. We denote $t_{\text{im}}, t_e, t_s, t_{\text{rm}}$ the respective average times spent on transmission of *im*, execution of M, system scheduling of this execution and transmission of *rm* respectively. The letters t and τ are used for the PC and for the server respectively.

Time in milliseconds	*Local execution*	*Remote execution*
Transmission of *im*	$t_{\text{im}} = 0$	$\tau_{\text{im}} = 10$
Execution	$t_e = 80$	$\tau_e = 10$
Scheduling	$t_s = 0$	$\tau_s = 50$
Transmission of *rm*	$t_{\text{rm}} = 0$	$\tau_{\text{rm}} = 10$
Total execution time	80	80

Table 1: Local vs. remote method execution of $O.M$

Such values can be easily obtained by simulation or by simply running the object locally on the user PC and also on processing servers. They can also be evaluated using similar techniques as those used for the evaluation of maximum time execution of real time application [11]. In the experiment the method example was run on a PC 486 DX2/33 with Unix SVR4 and a Sun Sparc server S690 with SunOS 4.1. The transmission time values reported in the table correspond to the average message time delivery between applications on the (heavy loaded) Ethernet network of workstations of our team for a message size of 1 kB (most object invocation messages are short and can be sent within such message).

In the situation depicted in this table, fragmentation is of interest because performance of the fragmented application is identical to the local execution of the non-fragmented application. The invocation is local on the PC therefore: $t_{\text{im}} = t_{\text{rm}} = 0$ s; moreover the PC is not shared and only the current application is active: therefore $t_s = 0$ s. We consider also that the mean transmission time for the invocation and reply messages is the same: $\tau_{\text{im}} = \tau_{\text{rm}} = \tau_m$. Therefore the total execution times t and τ of the local object invocation on a user PC and the remote object invocation on a processing server are:

$$\begin{cases} t = t_e \\ \tau = 2\tau_m + \tau_s + \tau_e \end{cases}$$

The relative response time is given by:

$$\tau - t = 2\tau_m + \tau_e + \tau_s - t_e$$

Running $O.M$ on the server is interesting if $\tau - t \leq 0$, i.e. if the remote execution is faster than the local one. Suppose now that the execution time on the server (including scheduling time) is proportional to the execution time on the PC: $\exists \lambda, \tau_e + \tau_s = \lambda t_e$. As the server is supposed more powerful than the PC we have $\lambda < 1$ and therefore:

$$\tau - t \leq 0 \iff t_e \geq \frac{2\tau_m}{1-\lambda}$$

For instance, going back to the simple example given, $\lambda = 0.6$ and t_e should be greater than 80 ms for $O.M$ being executed remotely. For simple methods with short execution time, and if we consider more powerful computers (several orders of magnitude as powerful as the user PC), the relative execution time obtained on the processing server (including the average scheduling time) becomes negligible, i.e. $1 - \lambda \simeq 1$ and therefore $\tau - t \leq 0 \iff t_e \geq 2\tau_m$: in this case, a non confidential object can be executed remotely provided that the execution time of the method measured on the PC is greater or equal to the round trip time of invocation/reply messages. Using high speed networks this round trip time should be much lower than some ms. Thus, objects whose average execution time of the methods is about a few hundreds of μs should be remotely executed without any degradation in performance.

When considering processing servers with a specific architecture (such as massively parallel computers) the implementation of an object (matrix computation and other complex numeric computations, image processing) can also be very efficient. This can be true not only because of the hardware architecture of the node but also according to available software tools and libraries. This strengthen the assumption $1 - \lambda \simeq 1$. On the other hand the designed objects can be implemented to be run in parallel on several nodes. For the above reasons, in some cases the scattered execution of the fragmented application is more efficient than the local execution.

5.2 Trade-offs between cost and performance

We examine in this section several solutions that could be investigated for running confidential applications and we concentrate on qualitative evaluation

[5] A Unix system with a single user for example.

Authorized licensed use limited to: ISAE. Downloaded on June 8, 2009 at 04:34 from IEEE Xplore. Restrictions apply.

aspects. The best solution in terms of performance (solution A) should be to only use secured non shared (personal) powerful processing servers. This solution is very costly, the CPU usage very low and it prevents using shared processing servers on which intrusions can be performed. The worse solution in terms of performance (solution C) is to only use personal low cost user PCs in a secure environment for running sensitive applications. The performance is very bad and, as in the previous solution, the execution of the application does not take advantage of existing processing servers. The fragmented solution (solution B) lies in between. This solution involves trusted personal user workstation at low cost and several (existing) untrusted powerful processing servers. This situation is illustrated on figure 7.

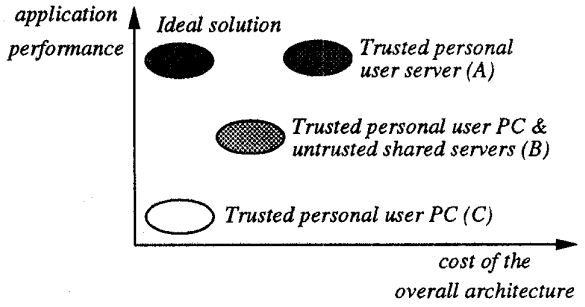

Figure 7: Trade-off between performance and cost

From a performance viewpoint solution B can provide better results than solution C as soon as parallelism among objects belonging to $\mathcal{NC}$ is considered and the implementation of specific objects is optimized. Solution B maximizes the use of shared (existing) computing power and reduces the cost of the overall architecture (see figure 8).

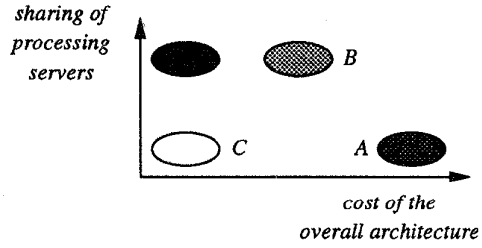

Figure 8: Trade-off between performance and sharing

Finally, solution B is of course more flexible since adding either trusted users PCs or processing servers can be done independently according to the needs in terms of users access to the system and in terms of computing power. It is also possible to run replicated copies of remote objects for fault tolerance purpose (using software-based techniques) without endangering the security of an application. Fault tolerance at the user station must be based on other solutions, e.g. based on stable storage. Fault tolerance is thus provided with standard computers instead of more costly specific fault tolerant computers. Finally, the flexibility of our solution may allow some degradation in terms of performance to be acceptable.

6 Conclusion

We have shown in this paper that the object oriented approach to application design enables confidentiality to be easily taken into account since objects represent real abstractions with a clear semantics; it is thus easy to decide whether an object is confidential or not. Looking more carefully to the question *why* is an object confidential often leads to perceive a confidential information (object) as a collection of non confidential items (sub-objects). Confidential objects can thus be substituted by a collection of sub-objects, some of which being non confidential. Links between sub-objects are kept within still confidential objects for which a trusted computing environment is required. Non-confidential objects can be executed on shared untrusted processing servers. Since the notion of object gathers both data and processing our solution provides security of data processing in distributed systems.

The performance aspects is one key aspect of objects fragmentation. A priori, object fragmentation can be done independently of performance aspects, since the aim is first to encapsulate confidential processing within few confidential objects. The recursion ends as soon as object substitution is not useful from a confidentiality viewpoint (Fragmentation). Performance aspects come into place when the placement of objects has to be decided (Scattering). Remote execution of non confidential object replicas (Redundancy) is sound when it does not lead to high degradation of performances; communication overheads (on high speed LANs) are then balanced by the high computing power of processing servers. The recent and future advances in network technology will make this assumption more and more realistic. As a side effect, simple objects should be executed remotely without any performance degradation. Moreover, considering parallelism between non confidential objects in the implementation of the application, but also for some particular application objects implemented on specific architectures, some gain in performance can be expected.

Authorized licensed use limited to: ISAE. Downloaded on June 8, 2009 at 04:34 from IEEE Xplore. Restrictions apply.

Finally, from a system architecture viewpoint, the proposed approach to the design of sensitive application provides more flexibility than other solutions. The system architecture can be organized as a set of low cost secured users non shared workstations and a set of high performance shared processing servers. The latter computers can be off-the-shelf computers without any specific features with respect to security or to fault-tolerance (just software-based mechanisms). The two types of computing units can be added independently according to the needs.

In the future, more and more confidential applications (even classified into different security levels) will be run in shared existing distributed computing architectures. Our approach provides a way to take advantage of the existing architectures for running confidential applications at low cost since just non shared users workstations must be logically and physically secured.

Acknowledgements

The authors wish to thank very much Brian Randell from the University of Newcastle-upon-Tyne (UK) who participated in the elaboration of these ideas during the numerous discussions on the subject.

References

[1] N. Ahituv, Y. Lapid, and S. Neumann, "Processing Encrypted Data", *Communications of the ACM*, vol. 30, no. 9, pp. 777–780, September 1987.

[2] R. L. Rivest, L. Adleman, and M. L. Dertouzos, "On Data Banks and Privacy Homomorphisms", in R. A. Demillo, D. D. Dobkin, A. K. Jones, and R. J. Lipton, editors, *Foundations of Secure Computation*, pp. 169–179. Academic Press, 1978, ISBN 0-12-210350-5.

[3] NCSC, "Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria", Technical Report NCSC-TG-005, National Computer Security Center, July 1987.

[4] Y. Deswarte, L. Blain, and J.-C. Fabre, "Intrusion Tolerance in Distributed Computing Systems", *in Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, pp. 110–121, Oakland (CA), USA, May 1991.

[5] J.-C. Fabre, Y. Deswarte, and B. Randell, "Designing secure and reliable applications using frs: An object-oriented approach", *in Proceedings of the First European Dependable Computing Conference (EDCC-1)*, pp. 21–38, Berlin, Germany, 1994. Springer-Verlag, Lecture Notes in Computer Science 852.

[6] R. Lea and J. Weightman, "Supporting Object-Oriented Languages in a Distributed Environment: The COOL Approach", *in Proceedings of the Fifth Technology of Object-Oriented Languages and Systems Conference*, pp. 37–47, Santa Barbara (CA), USA, 1991. Prentice Hall.

[7] J. G. Steiner, C. Neuman, and J. I. Schiller, "Kerberos: an authentication service for open network systems", *in Proceedings of the USENIX Winter Conference*, Dallas (TX), USA, Feb. 1988.

[8] B. Taylor and D. Goldberg, "Secure Networking in the Sun Environment", *in Proceedings of the USENIX Summer Conference*, pp. 28–37, Atlanta (GA), USA, 1986.

[9] A. Shamir, "How to share a secret", *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.

[10] M. O. Rabin, "Efficient information dispersal for security, load balancing and fault tolerance", *Journal of the ACM*, vol. 36, no. 2, pp. 335–348, Apr. 1989.

[11] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabl, C. Senft, and R. Zainlinger, "Distributed Fault-Tolerant Real-Time Systems: The MARS Approach", *IEEE Micro*, vol. 9, no. 1, pp. 25–40, February 1989.

Authorized licensed use limited to: ISAE. Downloaded on June 8, 2009 at 04:34 from IEEE Xplore. Restrictions apply.