## 10.6 Execution Domain

Unauthorized processes could destroy or modify information in files or databases, with obvious results, or could interfere with the execution of other processes. Therefore, define an execution environment for processes, indicating explicitly all the resources that a process can use during its execution, as well as the type of access to the resources.

### *Example*

In our operating system we know now how to assign access rights to processes and how to enforce these rights at execution time. However, a process may have different functions and in each functional mode it may need different rights. For example, if a process needs to read some files to collect some data, this should happen only at the specific time of access to the file, otherwise a hacker could take advantage of the extra rights to perform illegal accesses.

### *Context*

A process executes on behalf of a user, group, or role (a subject). A process must have access rights to use the resources defined for its subject during execution. The set of access rights given to a process define its execution domain. At times the process may also need to enter other domains to perform its work: for example, for example, to extract data from a file in another user's domain. Frequently, users structure their domains as a hierarchical tree of domains with one root domain.

### *Problem*

Restricting a process to a specific set of resources is a basic step towards controlling malicious behavior. Otherwise, unauthorized processes could destroy or modify information in files or databases, with obvious results, or could interfere with the execution of other processes.

The solution to this problem must resolve the following forces:

- There is a need to restrict the actions of a process during its execution; otherwise it could perform illegal actions.
- Resources typically include memory and I/O devices, but can also be system data structures and special instructions. Although resources are heterogeneous, we want to treat them uniformly.

- A process needs the flexibility to create multiple domains and to enter inner domains for specific purposes.
- There should be no restrictions on how to implement the domain.

## Solution

Attach a set of descriptors to the process that represent the rights of the process. Collect them into an execution domain. Execution domains can be nested.

## Structure

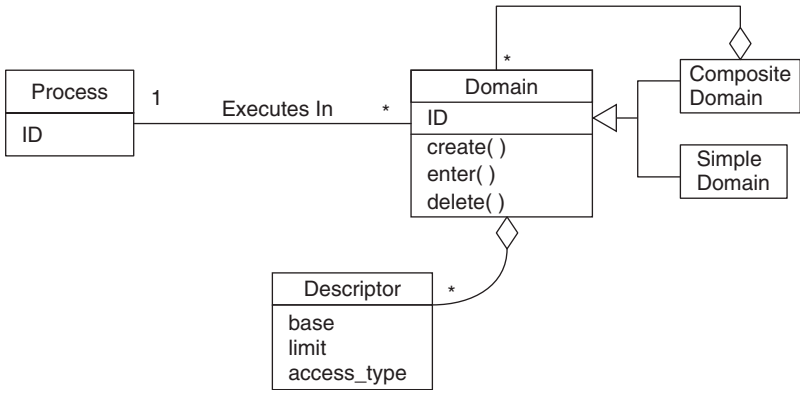
In the figure below, the `Domain` class represents domains, and, in conjunction with COMPOSITE, it describes nested domains. The operation `enter()` in `Domain` lets a process enter a new domain. A domain includes a set of descriptors that define rights for resources.

## Example Resolved

Using the concept of execution domain, we have a set of well-defined environments with explicit rights. A process can change domains according to its tasks and acquire only the needed rights in each domain.

## Known Uses

The concept of domains comes from Multics [Gra68]. Segments or pages (as in EROS [Sha02]) are structured as tree directories. The Plessey 250 and the IBM S/6000 running AIX [Cam90] are good examples of the use of this pattern. The Java Virtual Machine defines restricted execution environments in a similar way [Oak01].




Class diagram for EXECUTION DOMAIN

## Consequences

The following benefits may be expected from applying this pattern:

- The pattern lets users apply the principle of least privilege to processes— they can be given only the rights they need to perform their functions.
- It could be applied to describe access to any type of resource if the resource is mapped to a specific memory address.
- Processes may have several execution domains, either peer or nested.
- The model does not restrict the implementation of domains. A domain could be represented in many ways. For example, the Plessey 250, IBM S/38, and IBM S/6000 use capabilities. The Intel X86 and Pentium series and their corresponding operating systems use descriptors for memory access control.
- One can define special domains with predefined rights or types of rights. For example, Multics and the Intel X86 series use Protection Rings, in which each ring is assigned to a type of program, for example Supervisor, Utilities, User programs, and External programs. The rings are hierarchically structured based on their level of trust. Descriptors are used to cross rings in program calls.
- As shown, the descriptors refer to VAS segments, which is the most usual implementation. However, they could indicate resources not mapped to memory.

The following potential liabilities may arise from applying this pattern:

- Extra complexity—special hardware is needed to handle descriptors and set up domains.
- Performance overhead in setting up domains and in entering and leaving domains. Because of this, some operating systems for Intel processors use only two rings, improving performance but reducing security.
- Setting up the execution domain is implementation-dependent. In descriptor systems the operating system creates a descriptor segment with the required descriptors. In capability systems the descriptors are part of the process code and are enabled during execution.

## See Also

CONTROLLED PROCESS CREATOR (328) and CONTROLLED OBJECT MONITOR (335) work in conjunction with this pattern.