**Mantenimiento**

THE tyranny of the dominant decomposition [1] implies that, no matter how well a software system is decomposed into modular units, some functionality (often called a concern) crosscuts the decomposition. In other words, such functionality cannot be captured cleanly inside one single module and, consequently, its code will be spread throughout other modules.

From a maintenance point of view, such a crosscutting concern is problematic. Whenever this concern needs to be changed, a developer should identify the code that implements it. This may possibly require him to inspect many different modules, since the code may be scattered across several of them. Moreover, identifying the code specifically related to the relevant concern may be difficult. Apart from the fact that the developer may not be familiar with the source code, this code may also be tangled with code implementing other concerns, again due to crosscutting. It should thus come as no surprise that identifying crosscutting code may be a time-consuming and error-prone activity, as shown by Soloway et al. for delocalized plans [2].

**AO**

Aspect-oriented software development (AOSD) has been proposed for solving the problem of crosscutting concerns. Aspect-oriented programming languages have an abstraction mechanism targeted specifically at crosscutting concerns, called an aspect. This mechanism allows a developer to capture crosscutting concerns in a localized way.

**Pasaje de OO a AO**

In order to use this new feature and make the code more maintainable, existing applications written in ordinary programming languages could be evolved into aspect oriented applications. Once again, this requires identifying the crosscutting concern code such that it can be refactored using aspects. The activity of finding opportunities for the use of aspects in existing systems is typically referred to as aspect mining [3].

To support developers in these tasks, some form of automation is highly desirable.